

WEB

第 6 章 JSP 技 术

本章目录

- 6.1 JSP简介
- 6.2 JSP指令
- 6.3 JSP脚本程序
- 6.4 JSP动作
- 6.5 JSP内置对象
- 6.6 JSP开发实例
- 6.7 本章小结



引例

JSP (Java Server Pages) 是一种动态网页技术，该技术为创建显示动态生成内容的 Web 页面提供了一种简捷而快速的方法。

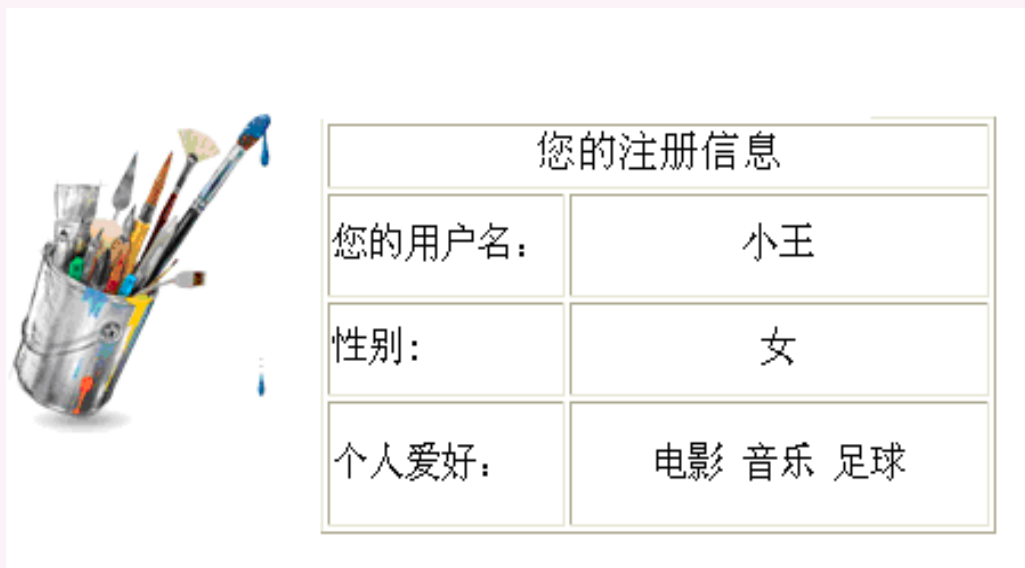


图 6-1 JSP 被服务器处理后返回的页面

虽然返回到每个用户的浏览器中的注册信息都是不同的，但是其实这些 HTML 页面都是由服务器执行同一个 JSP 产生的结果。

6.1 JSP 简介

JSP 也是 J2EE 组件技术之一，从 Web 应用的开发架构来看，JSP 属于 MVC（Model - View - Controller）架构中的视图层（View）组件，有**两个主要功能**：一是将用户表单的输入数据和请求传递给控制器层和模型层组件，另一个则是接受模型层处理完的数据显示给用户。

JSP 的工作原理

JSP 基本的工作原理与 Servlet 相似，但是在细节上 JSP 则较为复杂。在 JSP 第一次被用户请求时，JSP 文件将被容器的 JSP 引擎转换成为一个类似 Servlet 的 **Java 代码**。如果转换成功了，则所转换生成的 Servlet 代码继续被编译生成二进制字节码文件（**Class 文件**）。

下面以 MyEclipse 中一个 Web Project 的默认页面 index.jsp 为例（如图 6-2 所示），来看看 index.jsp 是怎么运行的。

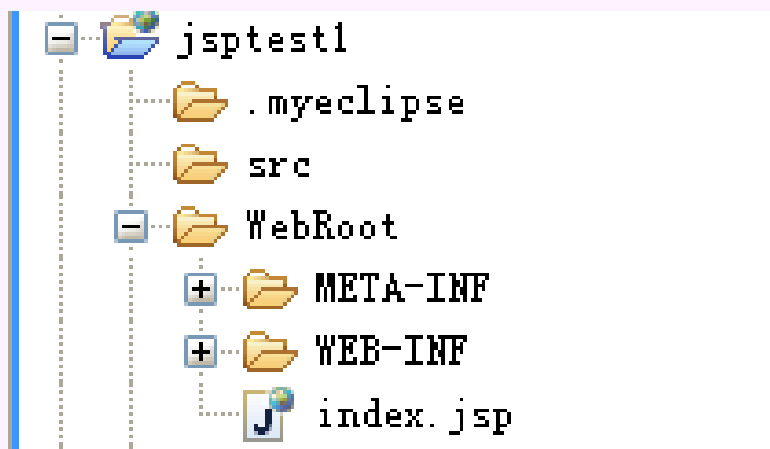




图 6-2 MyEclipse 中的 Web Project

将该项目部署到 Tomcat 的 webapps 目录下，做如下测试

- 1.在浏览器中输入 `http://localhost:8080/jsptest1/index.jsp`，并按回车发送请求，等待一段较长的时间后，浏览器中输出了“ This is my JSP page.”。
- 2.关闭浏览器再此访问。

JSP 的工作原理

在以上两个测试过程中，可以明显体验到第一次访问 `index.jsp` 的速度比后来访问的速度慢。我们来分析一下在刚才的过程中 Tomcat 究竟做了什么处理。打开 Tomcat 的 `work\Catalina\localhost` 目录，如图 6-3 所示。

名称	大小	类型
 <code>index_jsp.java</code>	4 KB	JAVA 文件
 <code>index_jsp.class</code>	5 KB	CLASS 文件

JSP 的工作原理

整个 JSP 的工作原理如图 6-5 所示。

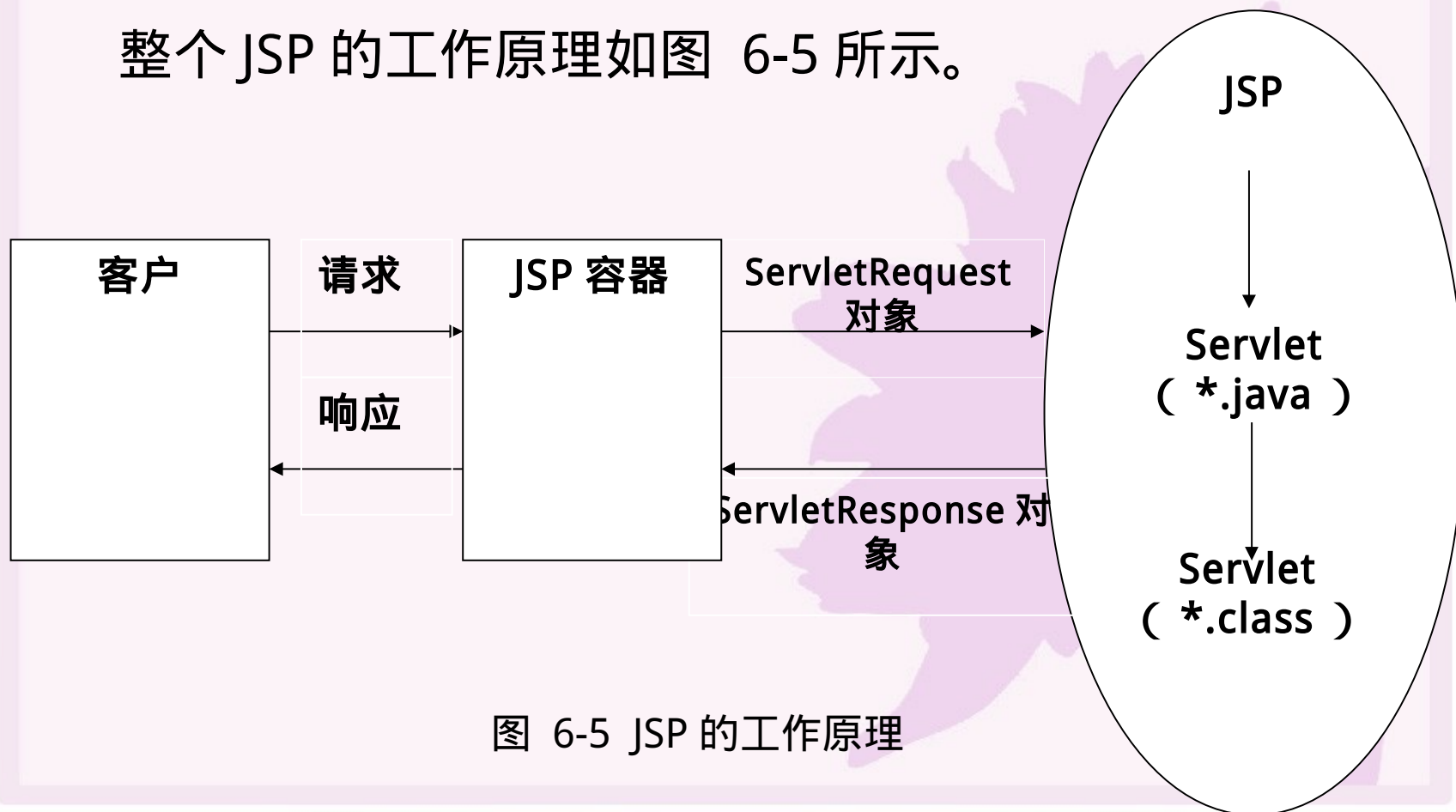


图 6-5 JSP 的工作原理

JSP 的特点

JSP 与静态网页 HTML 以及客户端动态脚本 JavaScript 之间的区别与联系：

1. 客户端动态网页和服务端动态网页都是以基本的 HTML 为基础，在 HTML 中分别嵌入 JavaScript 和 JSP 代码而构成的动态网页。

JSP 的特点

2. HTML 代码的解析由客户端浏览器完成，由浏览器显示其页面效果；客户端动态网页程序也是由浏览器执行，处理各种用户行为与浏览器、页面对象之间的交互事件；而服务器端动态网页则是由应用服务器执行，执行完后将处理的结果以 HTML 的形式返回给客户端浏览器。

3. HTML 和 JavaScript 代码的执行是直接被浏览浏览器解析执行，没有任何中间代码产生；而 JSP 页面则需要先被编译成为 Servlet 程序，并进一步被编译生成二进制字节码文件，最终被服务器的 Java 虚拟机所执行。
4. HTML 和客户端动态网页的源代码都可以在客户端被查看，而由于服务器端动态网页的执行在服务器端，因此客户端得到的结果中并不包含原始的服务器端动态代码，所以服务器端动态网页的源代码是比较安全的，只有在服务器端才能被查看。

6.2 JSP 指令

JSP 指令主要用来提供整个 JSP 网页相关的信息，并且用来设定 JSP 页面的相关属性。

JSP 指令的一般语法形式为：

`<%@ 指令名 属性 =" 值 " %>`

常用指令：

page
include

page 指令

page 指令定义 JSP 文件中的全局属性。它的作用范围是整个 JSP 页面。

page 指令具有以下常用属性：

(1) language : 该属性用于指定当前 JSP 页面将使用的语言。例如：`<%@ page language="java" %>`

(2) import : 该属性用于指定当前页面需要导入的包或类，与 Java 程序中的 import 语句类似。如果需要导入多个包或类，则用逗号隔开。例如：

`<%@ page import="java.util.*,java.sql.*" %>`

(3) contentType : 该属性用于指定向客户端输出内容的类型。例如 : `<%@ page contentType = "text/html;charset=gb2312" %>`

(4) errorPage : 该属性用于指定一个 JSP 作为专门的错误处理页来处理当前页面所有抛出的未被处理的意外错误。例如 :

`<%@ page errorPage = "errorPage.jsp" %>`

(5) isErrorPage : 该属性用于指定当前页面是否可以处理来自其它页面的错误 , 缺省为 "false" 。例如 : `<%@ page isErrorPage = "true" %>`

include 指令

include 指令的作用是包含另一个文件，其语法相当简单：

```
<%@ include file="..." %>
```

以下分别是 index.jsp 和 product.jsp 的页面结构图，如图 6-6 和图 6-7 所示：

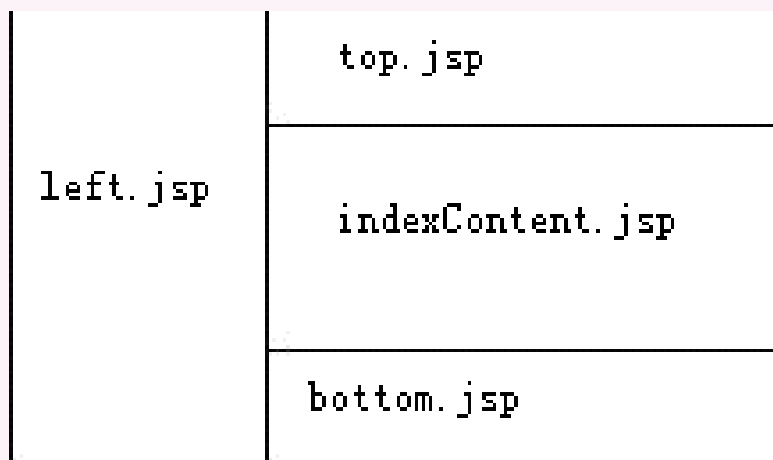


图 6-6 index.jsp 的页面结构

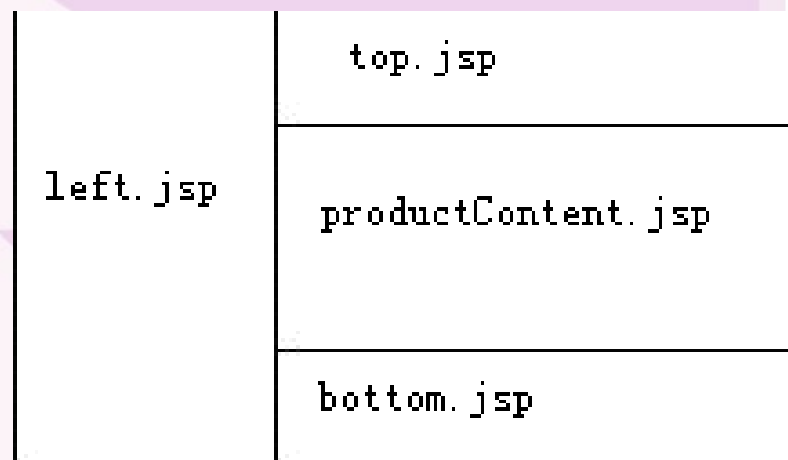
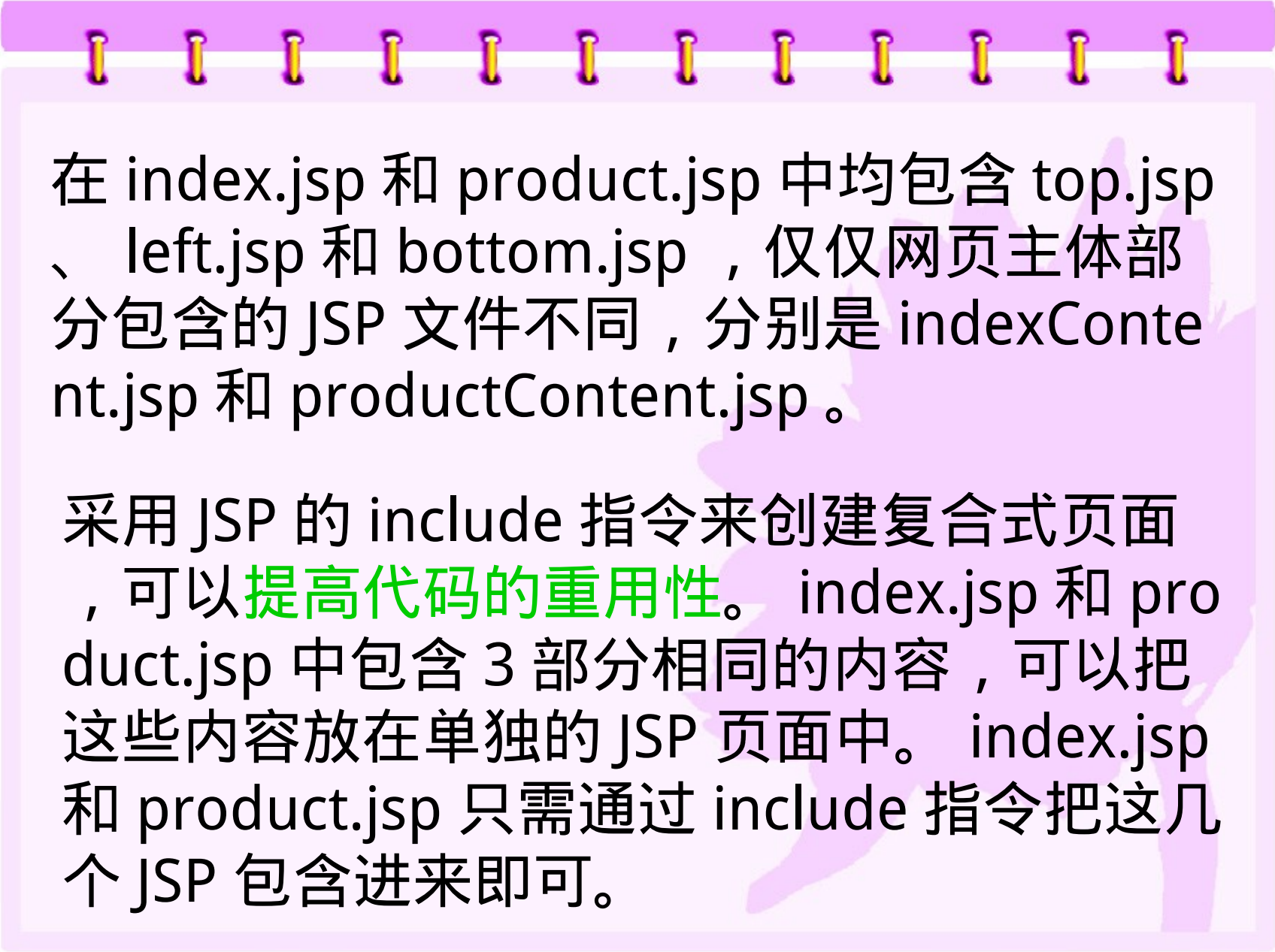


图 6-7 product.jsp 的页面结构



在 index.jsp 和 product.jsp 中均包含 top.jsp、left.jsp 和 bottom.jsp，仅仅网页主体部分包含的 JSP 文件不同，分别是 indexContent.jsp 和 productContent.jsp。

采用 JSP 的 include 指令来创建复合式页面，可以**提高代码的重用性**。index.jsp 和 product.jsp 中包含 3 部分相同的内容，可以把这些内容放在单独的 JSP 页面中。index.jsp 和 product.jsp 只需通过 include 指令把这几个 JSP 包含进来即可。

6.3 JSP 脚本程序

JSP 中主要的程序就是脚本程序，其中包括三个部分：JSP 声明（Declaration）、JSP 表达式 Expression）和 JSP 代码段（Scriptlet）。

从功能上讲，JSP 声明用于声明一个或多个变量，JSP 表达式是一个完整的语言表达式，而 JSP 代码段则是一些 Java 程序片断。三种脚本程序的基本语法都是以“<%”开头、“%>”结尾的。

JSP 声明

JSP 声明是在当前页面范围内声明合法的变量或方法，这些变量或方法也仅在当前页面中有效。语法如下：

```
<%! declaration; %>
```

或者

```
<% declaration; %>
```

以上是两种声明变量或方法的方式，区别在于是否在前面加“！”进行声明

举例如下：

```
<%! int i = 0; %>
```

```
<% int j = 0; %>
```

假设我们把以上的 JSP 声明嵌入到 index.jsp 中，然后打开浏览器访问 index.jsp，最后打开 Tomcat 的 work\Catalina\localhost 目录下的 index_jsp.java，我们可以看到，当前变量 i 已经成为这个 Servlet 类的成员变量，而变量 j 则出现在 _jspService 方法中。

我们可以分析得出：加“！”声明的变量其实就是该 JSP 生成的 Servlet 类的成员变量，属于全局变量，它的作用域是整个 Servlet 对象，只要 Servlet 对象在 Tomcat 容器中产生后，该变量就随着 Servlet 对象的存在而存在，直到 Servlet 对象被容器所释放；而不加“！”声明的变量则是该 JSP 生成的 Servlet 类中 _jspService 方法的局部变量，它的作用域仅仅是 _jspService 方法体，这个方法一旦被调用结束，该变量的生命周期也就结束了。

JSP 表达式

JSP 表达式包含一个符合 Java 语法的表达式。语法如下：`<%= expression %>`

JSP 表达式在运行后被自动转化为字符串，然后输出到这个表达式在 JSP 文件中所在的位置。

比如：

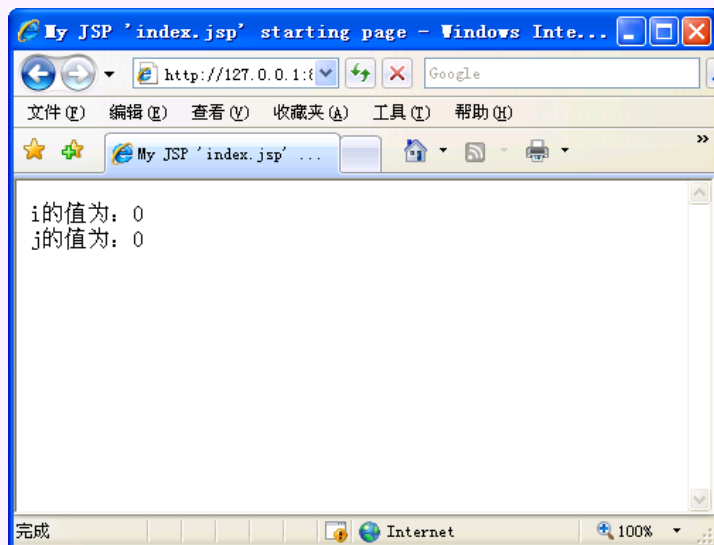
```
<%! int i = 0; %>
```

```
<% int j = 0; %>
```

```
i 的值为： <%=i++%><br>
```

```
j 的值为： <%=j++%>
```

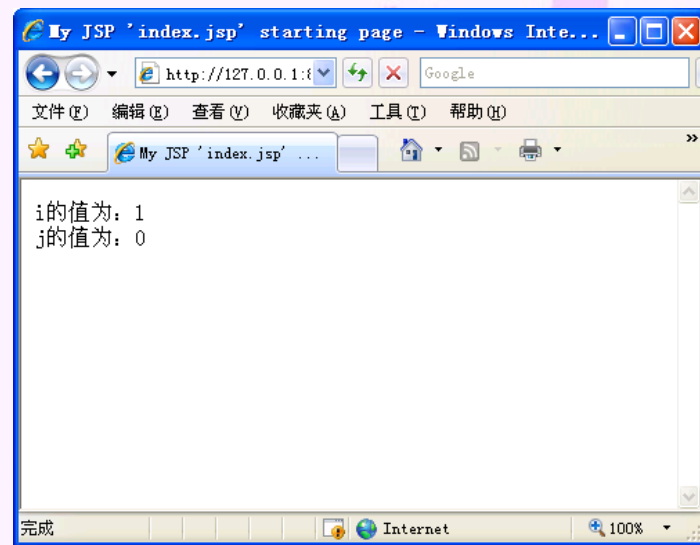
假设我们把以上的 JSP 声明和 JSP 表达式嵌入到 index.jsp 的 `<body></body>` 中，然后打开浏览器访问 index.jsp，得到结果如下图所示：



第一次访问：

i 的值为：0

j 的值为：0



第二次访问：

i 的值为：1

j 的值为：0

- 在首次访问 index.jsp 后，返回的结果 i 和 j 的值都为 0，而第二次访问后，返回的结果 i 的值为 1，j 的值仍然为 0，这样的现象可以用前面介绍的 JSP 工作原理和 JSP 声明来解释。
- 由于 i 用“!”声明，j 没有用“!”声明，因此，i 为当前 JSP 编译生成的 Servlet 对象的全局变量，而 j 只是 _jspService 方法中的局部变量。当客户端第一次访问 index.jsp 时，Tomcat 容器创建该 JSP 对应的 Servlet 的对象后，把 i 和 j 的值返回给客户端，再对 i 和 j 做后缀加法（++），i 和 j 的值都为 1。

由于 `i` 是该 Servlet 对象的成员变量，该 Servlet 对象从创建以后就一直在容器中，因此 `i` 的状态也随着 Servlet 对象的存在而延续；由于 `j` 是 `_jspService` 方法中的局部变量，它的作用范围仅在该方法体内部，虽然首次执行后 `j` 的值通过后缀加已经变为 1，但是该方法执行完后，`j` 的生命周期也就结束了，因此当客户端第二次访问该对象的时候，`_jspService` 方法重新被执行，而每次执行 `_jspService` 方法时，方法体内部的局部变量 `j` 都是重新声明的新变量，每次执行都是先输出“0”，然后做后缀加法。

核心代码如下：

```
public final class index_jsp extends  
org.apache.jasper.runtime.HttpJspBase
```

```
implements
```

```
org.apache.jasper.runtime.JspSourceDependent {
```

```
    int i = 0; // i 被声明在 _jspService 等方法外，属于全局变量
```

```
.....
```

```
.....
```

```
    public void _jspService(HttpServletRequest request,  
    HttpServletResponse response)
```

```
throws java.io.IOException, ServletException {
```

```
.....
```

```
    int j = 0; // j 被声明在 _jspService 方法内，属于局部变量
```

```
.....
```

```
}
```

JSP 代码段

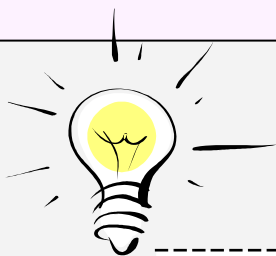
在网页中嵌入 Java 代码，这些嵌入后的代码就称为 JSP 代码段，又叫做“scriptlet”。

JSP 代码段的具体编写规范是：如果是 Java 代码段，必须放置于 `<%.....%>` 之间；如果出现了 HTML 的标记或文字，必须把这些内容放置在 `<%.....%>` 之外。基于这样的规范，通常我们所写的 JSP 代码段将会是由 Java 代码和 HTML 交叉混合的代码体。举例如下：

```
<%  
    String gender="female"; if(gender.equals("female")){  
    %>  
        She is a girl.  
    <% }else{ %>  
        He is a boy.  
    <% } %>
```

以上代码等价于该 JSP 编译生成的 Servlet 中的如下代码：

```
public void _jspService(HttpServletRequest request,
HttpServletRequest response)
throws java.io.IOException, ServletException {
.....
    String gender="female";
    if(gender.equals("female")){
        out.write(" \r\n");
        out.write("      She is a girl. \r\n");
    }else{
        out.write(" \r\n");
        out.write("\t He is a boy. \r\n");
    }
.....
}
}
```



小提示：

JSP 表达式的末尾是没有分号的，这点和本节中的 JSP 声明、JSP 代码不同，在 JSP 声明和 JSP 代码中，每个声明或每行代码的结尾都要使用分号结尾，否则当前 JSP 被编译时就会报语法错误。

JSP 动作

JSP 动作利用形如 XML 格式的标签来控制 JSP 引擎的行为，利用 JSP 动作可以实现动态包含网页文件、请求转发等功能。语法如下：

```
<jsp:XXX 属性 =" 属性值 " />
```

JSP 动作也可以叫做 JSP 标签，JSP 标签以“jsp”为前缀。在 JSP 页面被编译为 Servlet 文件期间，当容器遇到这个标记时，就用相应的 Java 代码来代替它。

include 动作

<jsp:include> 标签表示包含一个静态的或者动态的网页文件。语法如下：

```
<jsp:include page="path" flush="true" />
```

前面 JSP 指令中，我们已经介绍过 include 指令，它是在 JSP 文件被转换成 Servlet 时引入包含文件，而这里的 <jsp:include> 动作则不同，引入包含文件的时间是在页面被请求的时候，<jsp:include> 动作的文件引入时间决定了它的效率要稍微差一点。

forward 动作

<jsp:forward> 标签表示把当前流程控制转发到另一个网页文件。语法如下：

```
<jsp:forward page="path" />
```

这个动作将结束对当前页面的处理，而开始处理由 page 属性所指定的页面，整个转发过程中共用一个请求。

例子：以 get 方式向 forward1.jsp 发送请求传入参数，再以 forward 方式转发给 forward2.jsp, 由 forward2.jsp 取出参数。

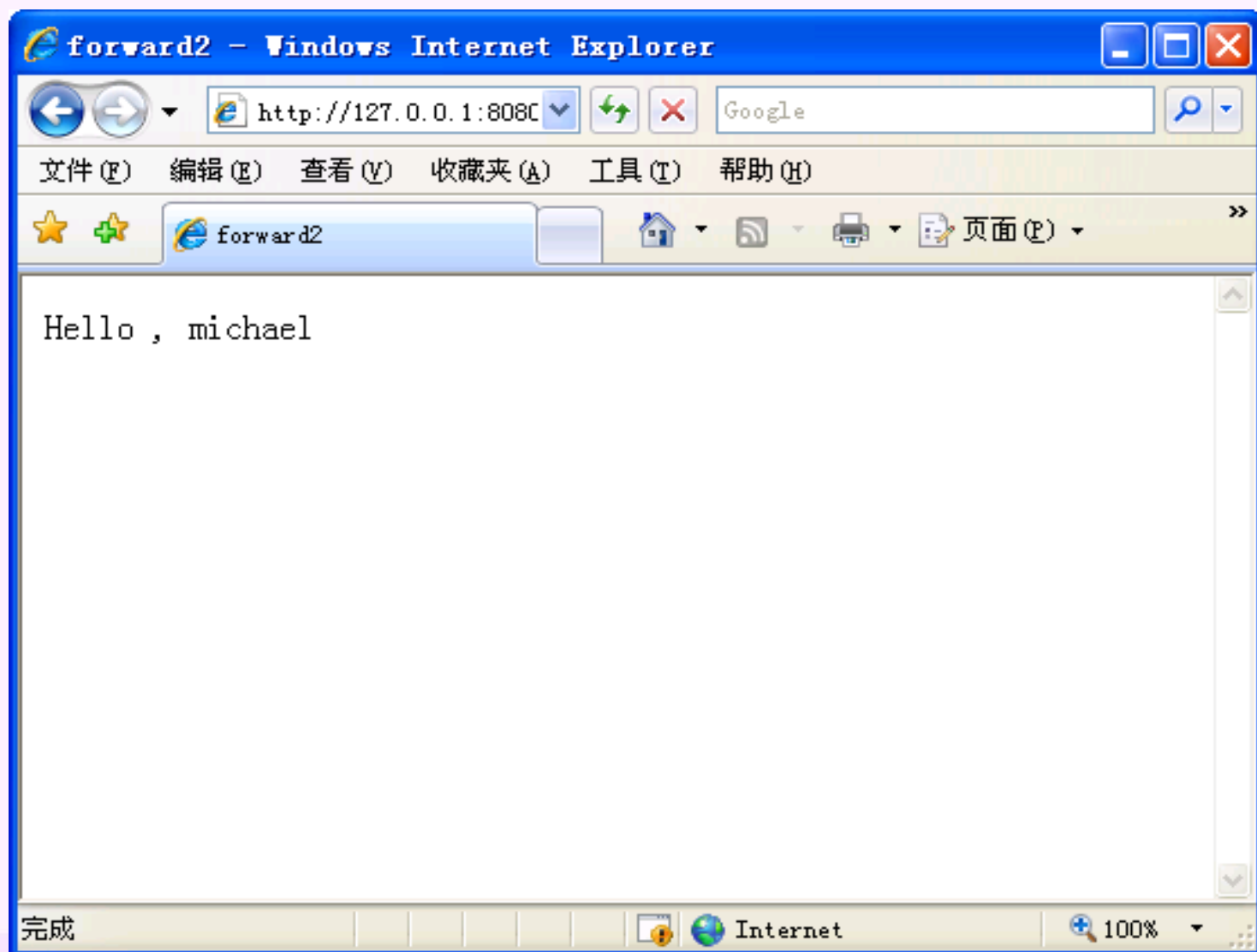
forward1.jsp 的代码如下：

```
<HTML>
<HEAD>
<TITLE>forward1</TITLE>
</HEAD>
<body>
    <jsp:forward page="forward2.jsp" />
</body>
</html>
```

forward2.jsp 的代码如下：

```
<HTML>
<HEAD>
<TITLE>forward2</TITLE>
</HEAD>
<body>
    Hello , <%=request.getParameter("username")
%>
</body>
</html>
```

在地址栏中输入“ <http://127.0.0.1:8080/forwardtest/forward1.jsp?username=michael>” ,
得到最后的结果如图 6-10 所示:



6.5 JSP 内置对象

JSP 基于 Java 语言，面向对象也是它的一大特色。JSP 中包含大量的内置对象，其中有 6 个常用的内置对象，这 6 个对象分别是：

out

request

response

session

application

page

这些对象不用声明就可以在 JSP 中直接使用。

out 对象

out 对象是 `javax.servlet.jsp.JspWriter` 的实例。out 对象一般只在 JSP 代码段内使用，它的主要功能是把 HTML 的内容输出到网页中。

表 6-1 out 对象的方法

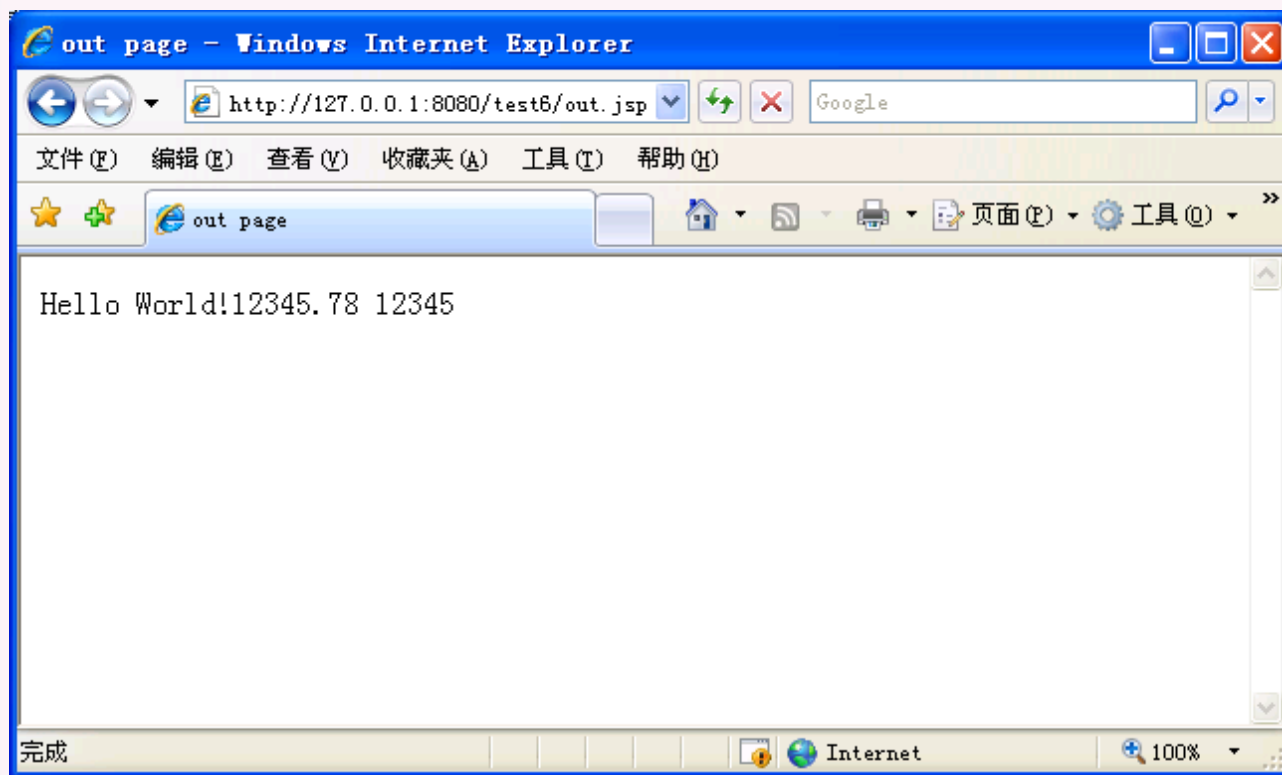
方法名	功能
print	输出各种类型的数据，不换行
newLine	输出一个换行符号
println	输出各种类型的数据，输出完毕后，进行换行
close	关闭输出流，终止当前页面的剩余部分输出

out 对象的使用示例：用 out 对象输出多种类型的数据

。

```
<%@ page language="java"
contentType="text/html;charset=gb2312"%>
<html>
<head><title>out page</title></head>
<body>
<%
out.print("Hello World!");
out.println(12345.78);
out.println(12345);
%>
</body>
</html>
```

以上代码中分别用 out 对象的 print 方法和 println 方法输出字符串类型、double 类型以及 int 类型的数据，访问得到的结果如图 6-11 所示：



为什么调用了 println 方法并没有起到换行的效果？

在当前浏览器中点击右键菜单查看源文件，我们可以看到当前客户端得到的 HTML 的内容，如图 6-12 所示：



```
<html>
<head><title>out page</title></head>
<body>
Hello World!12345.78
12345

</body>
</html>|
```

原来 println 方法的换行指的是在 HTML 源文件中输出后的“换行”，而并不是页面被浏览器解析后呈现出来的“换行”

request 对象

request 对象是 `javax.servlet.http.HttpServletRequest` 的实例。

当客户端请求一个 JSP 网页时，Tomcat 服务器会将客户端的请求信息包装在这个 request 对象中，请求信息的内容包括参数名、参数值以及客户端的主机地址、名称、端口等，然后我们可以通过这个 request 对象来取得上述有关客户端的信息。

表 6-2 request 对象的方法

方法名	功能
getServerName	返回接收请求的服务器的主机名
getRemoteAddr	返回发送请求的接口程序的 IP 地址
getServerPort	返回接收请求的端口
getMethod	返回请求所使用的方法
getParameter	返回包含指定参数的单独值的字符串
getParameterValues	返回作为字符串列举的指定参数的值
setAttribute	设置请求中属性的值
getAttribute	返回请求中属性的值
getCookies	返回客户端的 Cookie 信息

1.用 request 对象来获取服务器和客户端的各种信息。

<body>

服务器的 IP :

<%=request.getServerName()%>

服务器的端口 :

<%=request.getServerPort()%>

客户端 IP 地址 :

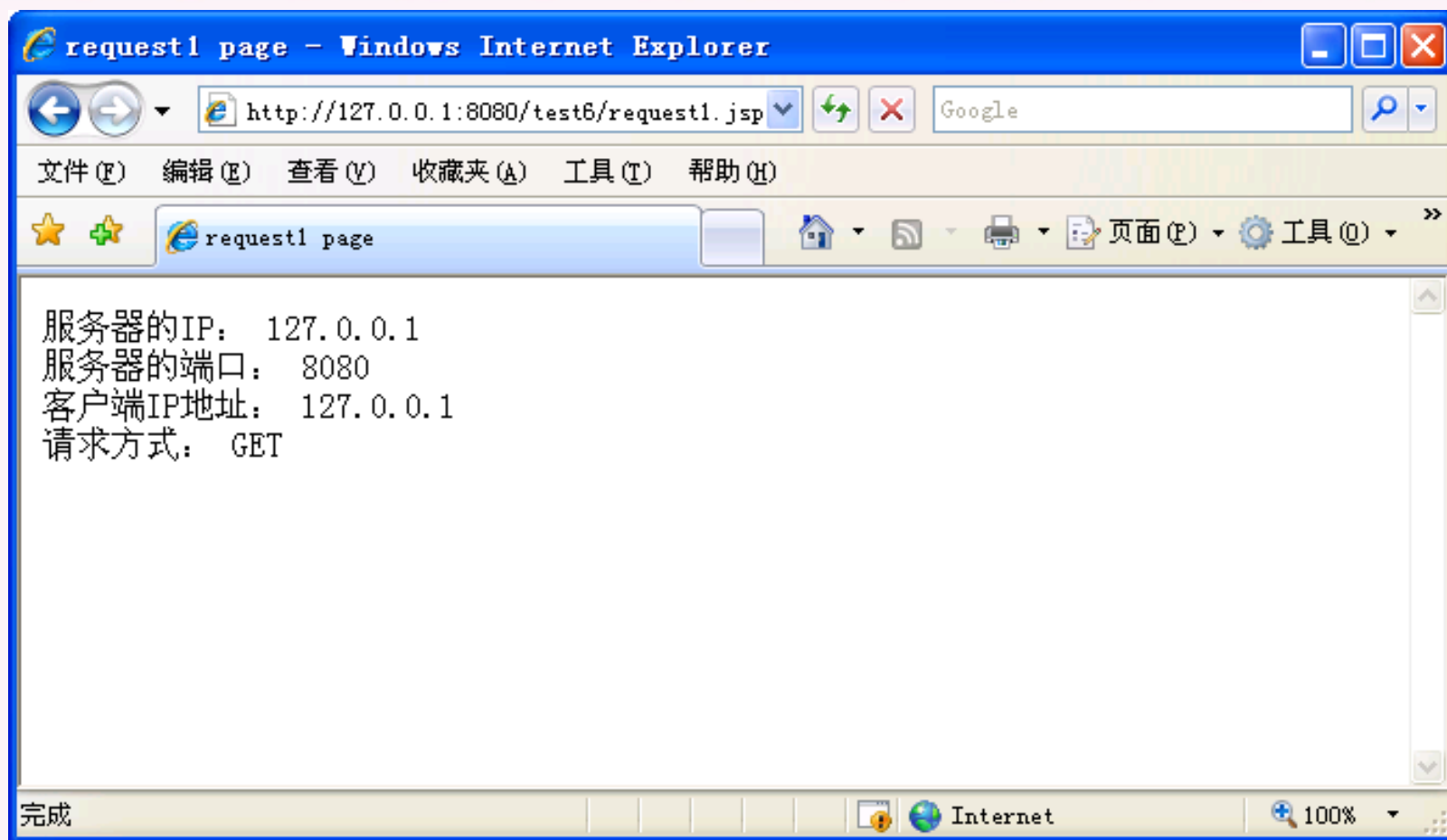
<%=request.getRemoteAddr()%>

请求方式 :

<%=request.getMethod()%>

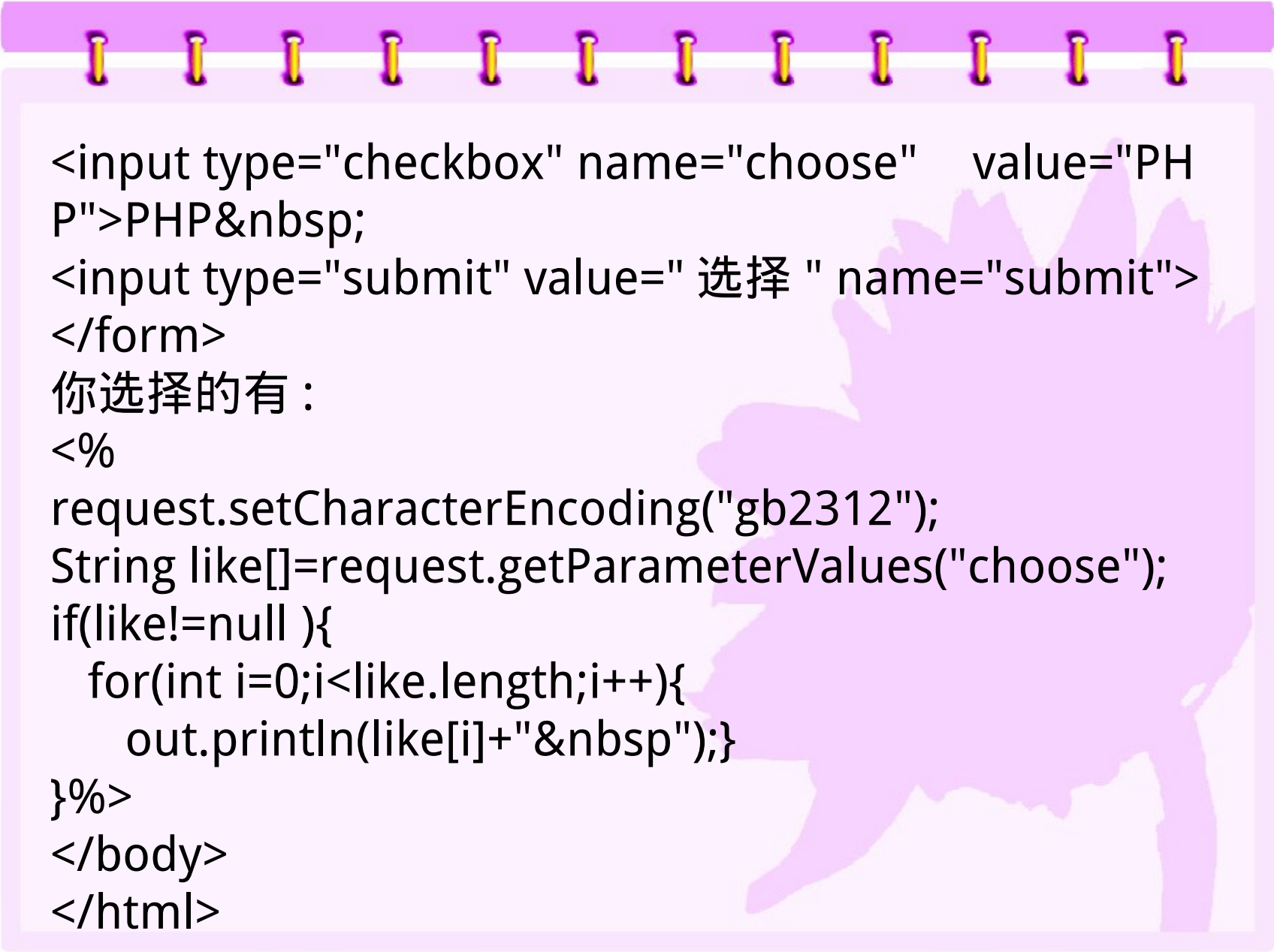
</body>

在地址栏中访问 request1.jsp 后得到的结果如图 6-13 所示



2. 用 request 对象来获取表单提交的参数值。

```
<%@ page language="java"
        contentType="text/html; charset=gb2312"%>
<html>
<head><title>request2 page</title></head>
<body><form action="request2.jsp" method="post">
动态网页有：
<input type="checkbox" name="choose" value="ASP">
ASP&nbsp;
<input type="checkbox" name="choose" value="ASP.N
ET">ASP.NET&nbsp;
<input type="checkbox" name="choose" value="JSP">J
SP&nbsp;
```

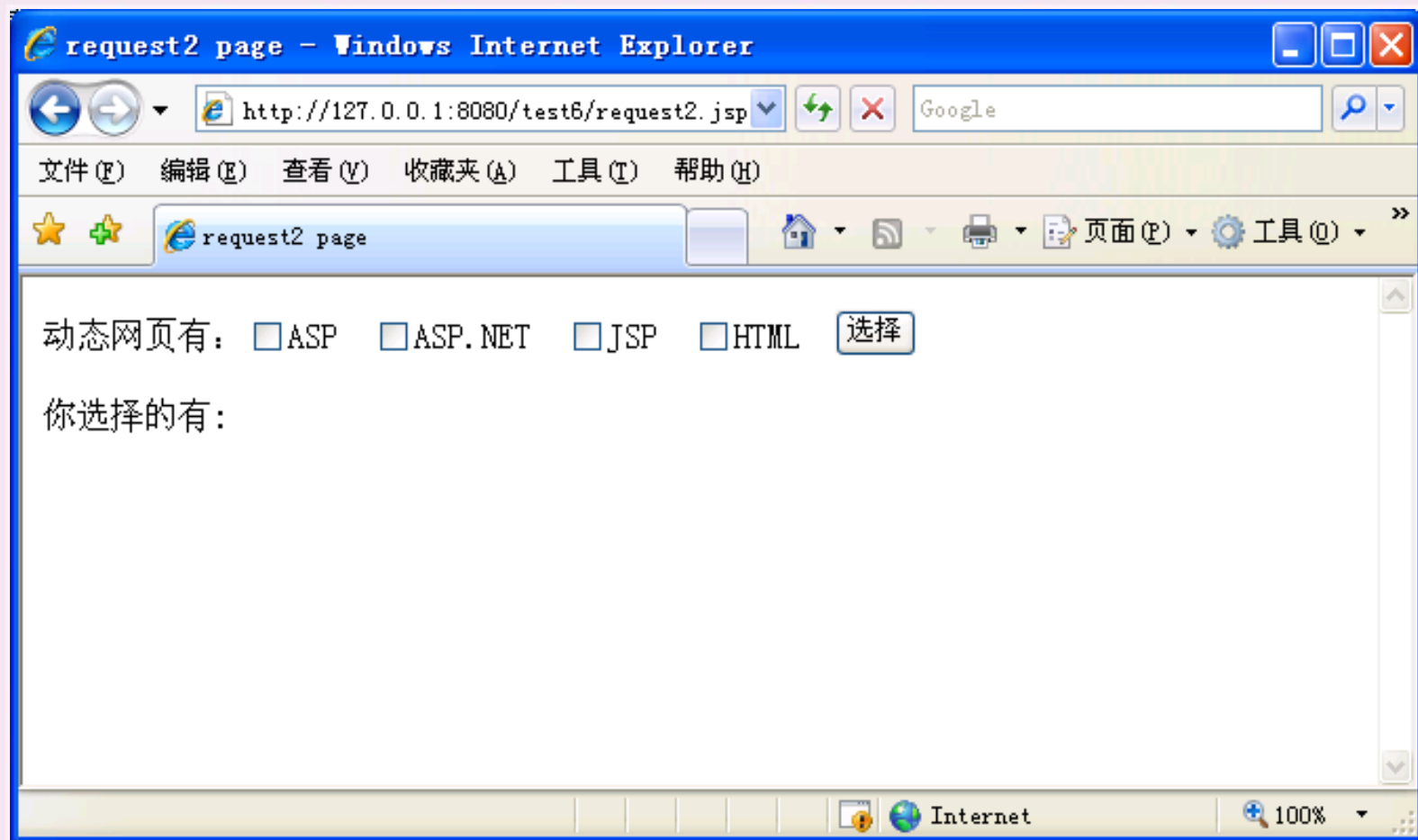


```
<input type="checkbox" name="choose" value="PHP">PHP   
<input type="submit" value=" 选择 " name="submit">  
</form>
```

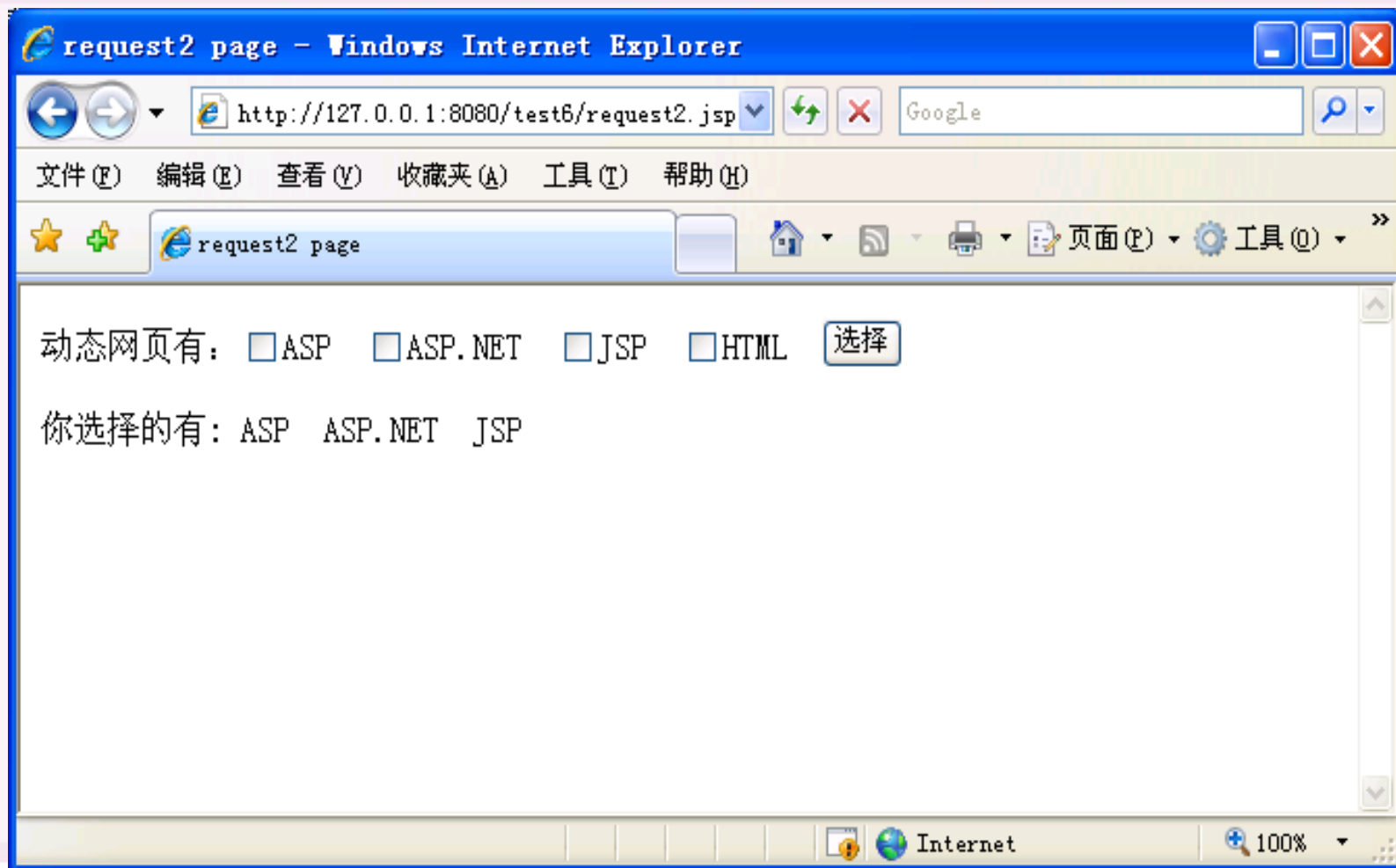
你选择的有：

```
<%  
request.setCharacterEncoding("gb2312");  
String like[]=request.getParameterValues("choose");  
if(like!=null ){  
    for(int i=0;i<like.length;i++){  
        out.println(like[i]+"&nbsp;");  
    }  
}%>  
</body>  
</html>
```

在地址栏中访问 request2.jsp 后得到的结果如图 6-14 所示：



选择其中的“ASP”、“ASP.NET”和“JSP”后，点击“选择”，显示的结果是“你选择的有：ASP ASP.NET JSP”，如图 6-15 所示：





小提示：

只有对于表单 POST 提交的参数才可以用 `request.setCharacterEncoding("gb2312")` 进行中文处理，其它情况只能用上一章中所介绍的方式——`new String(tempStr.getBytes("iso-8859-1"), "gb2312")`。

3. 用 request 对象获取客户端的 Cookie 信息。

Cookie 是当用户浏览某网站时，网站服务器存储到客户机上的一个小文本信息，它可以记录用户的会话 ID 以及登录名、密码等信息，当用户再次来到该网站时，网站通过读取该用户的 Cookie，便可以得知用户曾经访问该网站的相关信息

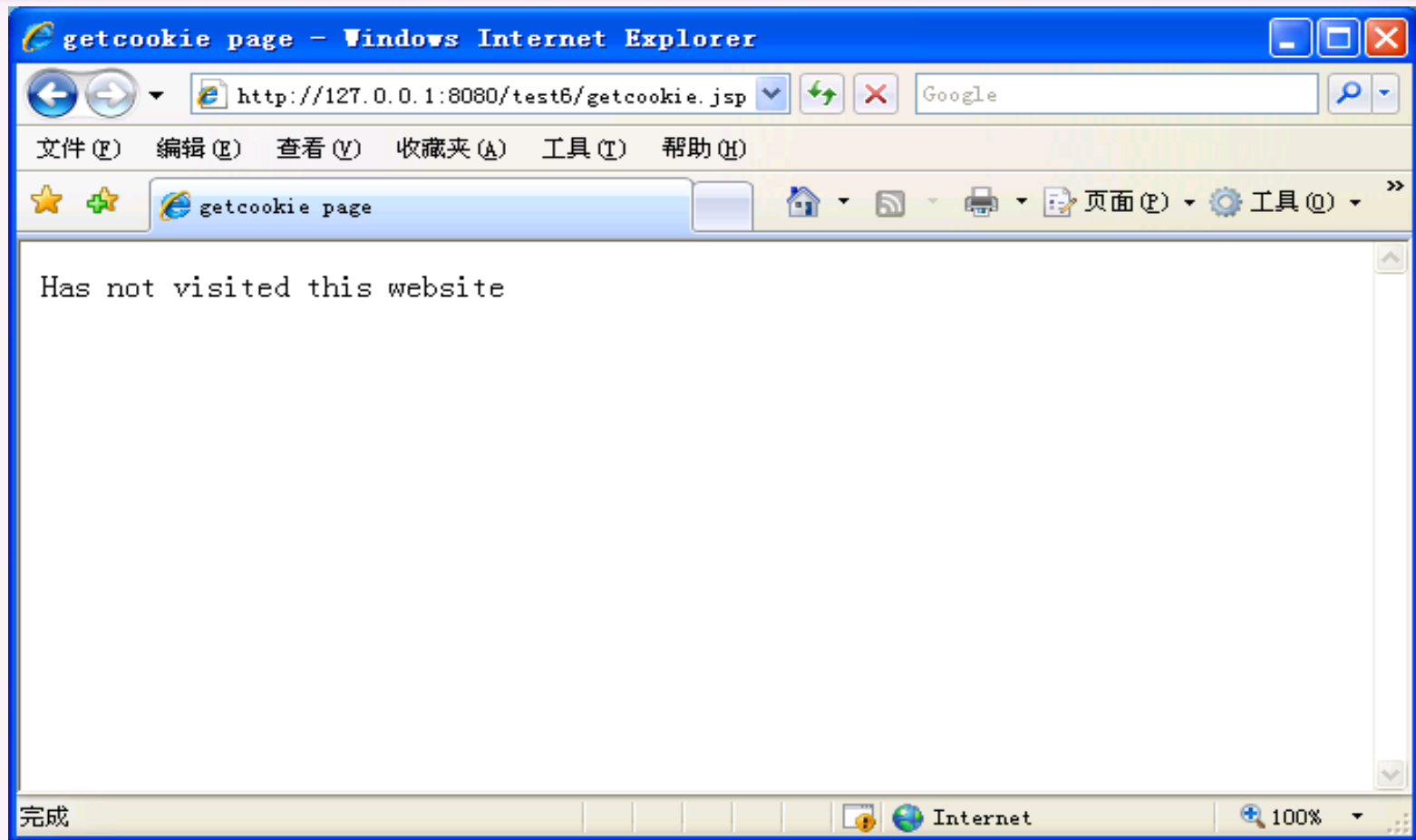
Cookie 默认保存在客户机的内存里，如果设置了 Cookie 的过期时间，浏览器就会把 Cookie 保存到硬盘上，关闭后再次打开浏览器访问该网站，这些 Cookie 仍然有效，直到超过设定的过期时间。

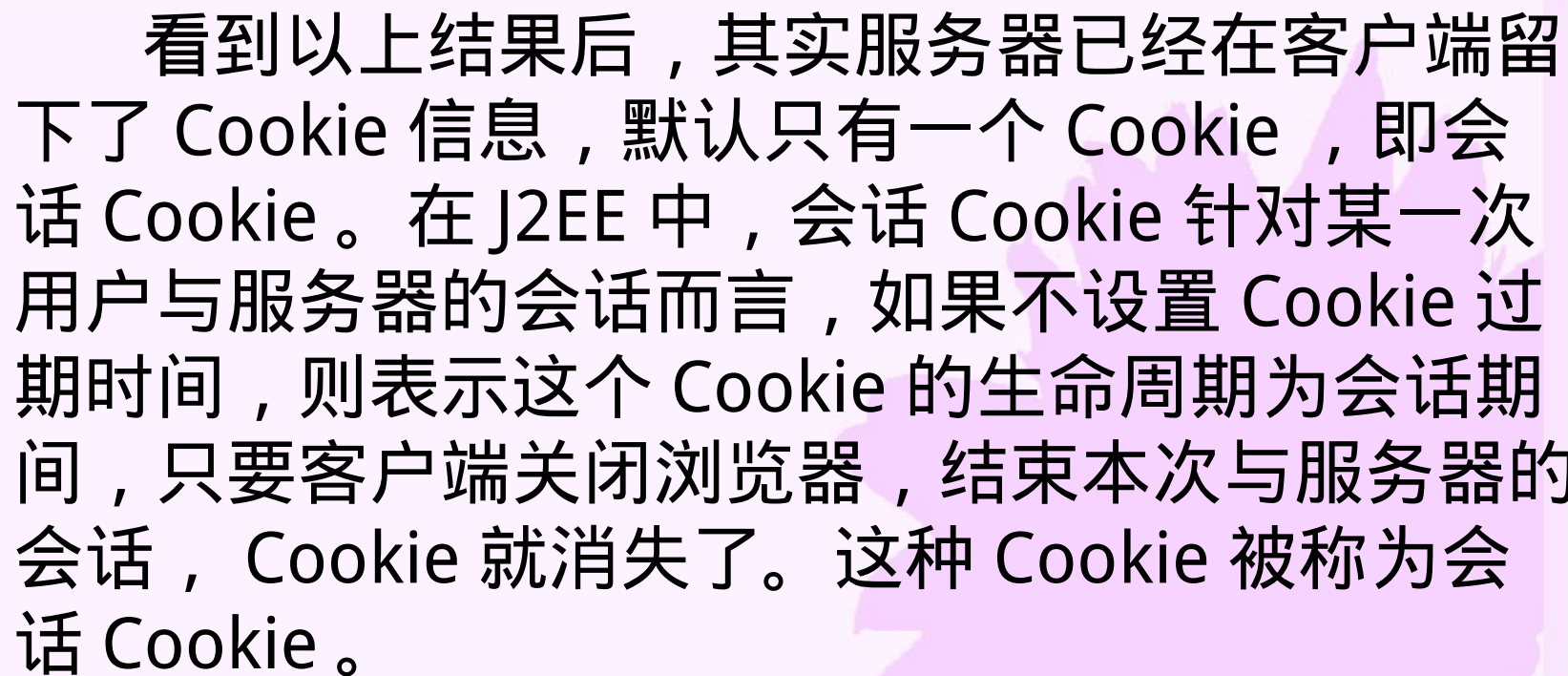
Java 中，实现 Cookie 对象的类是 javax.servlet.http.Cookie

JSP 内置对象 request 中有 getCookies 方法，返回的是 Cookie 对象数组。此外，response 对象有 addCookie 方法，可以将服务器端自定义的 Cookie 对象向客户机写入。

```
<% Cookie[] cookies = request.getCookies();  
if(cookies==null){  
    out.println("Has not visited this website");  
}else{  
    for (int i = 0; i < cookies.length; i++) {  
        out.println("cookie name:" + cookies[i].getName()+  
            "<br>");  
        out.println("cookie value:" +cookies[i].getValue());  
    }  
} %>
```

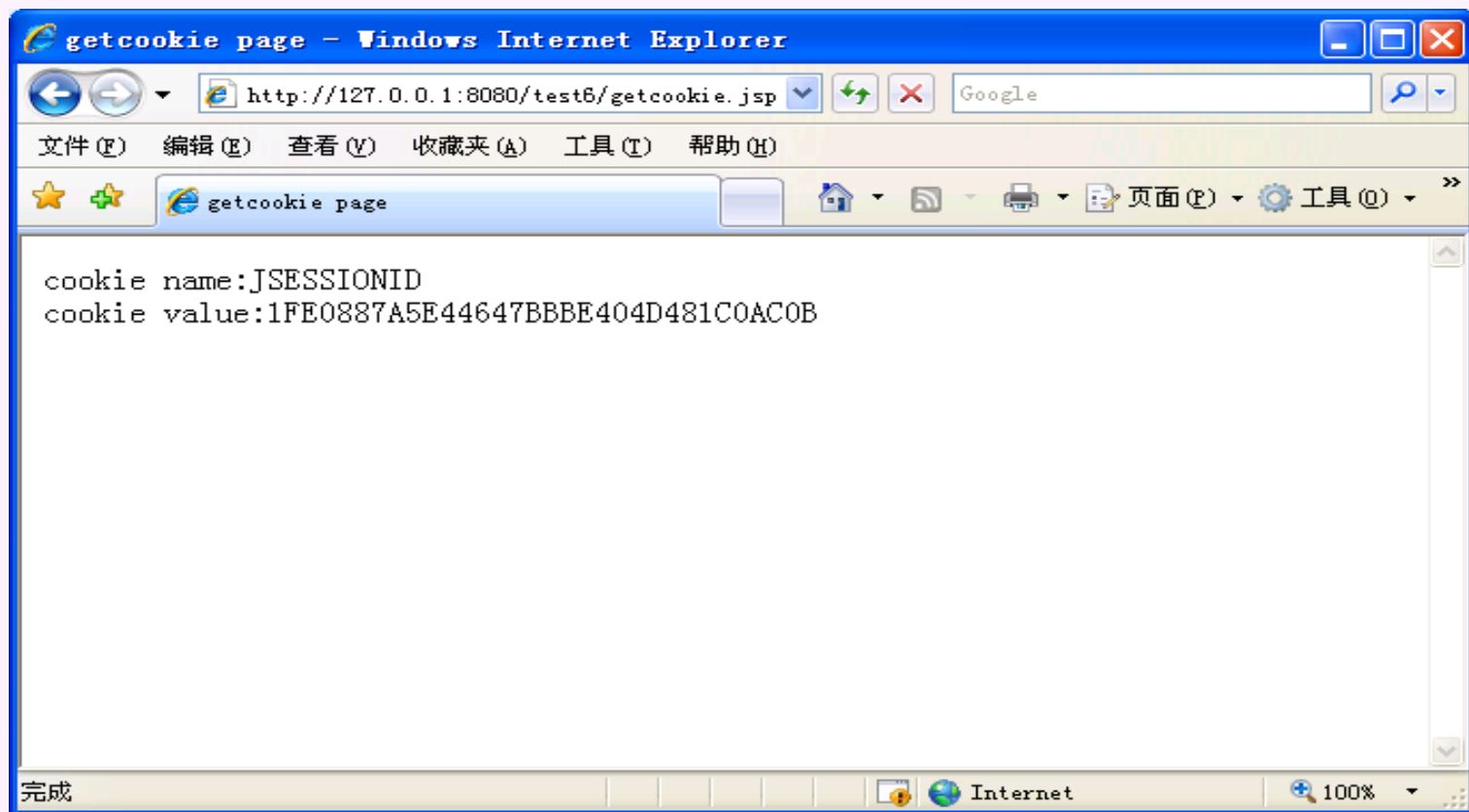
首先用 request.getCookies() 取出用户本机所有访问该网站留下的 Cookie 信息，如果本次是第一次访问该网站，则 Cookie 对象为空，显示的结果如图 6-16 所示：





看到以上结果后，其实服务器已经在客户端留下了 Cookie 信息，默认只有一个 Cookie，即会话 Cookie。在 J2EE 中，会话 Cookie 针对某一次用户与服务器的会话而言，如果不设置 Cookie 过期时间，则表示这个 Cookie 的生命周期为会话期间，只要客户端关闭浏览器，结束本次与服务器的会话，Cookie 就消失了。这种 Cookie 被称为会话 Cookie。

当用户再次刷新 `cookie.jsp`，此时 `request.getCookies()` 取出的 Cookie 不再为空，而是用户上次访问该页面留下的会话 Cookie，因此可以用循环的方式遍历 Cookie 数组



会话 Cookie 对象的名称为 JSESSIONID ，值为服务器生成的具有唯一性的字符串。

response 对象

response 对象是 `javax.servlet.http.HttpServletResponse` 的实例，Tomcat 服务器会根据客户端的请求建立一个默认的 response 对象，它通常用于将服务器端的处理结果返回给客户端的浏览器或进行重定向等操作。

表 6-3 response 对象常用方法

方法名	功能
<code>sendRedirect</code>	重定向客户端的地址
<code>addCookie</code>	添加一个 Cookie 对象，用来保存客户端的用户信息

用 response 对象实现页面重定向功能

response.jsp

```
<body>
<%
if(request.getParameter("username").equals("admin")){
    response.sendRedirect("index.jsp");
}
%>
</body>
```

这里的“response 重定向”与前面所介绍的“`<jsp:forward>` 转发”是不是实现完全一样的功能？答案是否定的。

重定向最大的一个特点就是：结束本次用户的请求，发送新的请求给下一个页面，因为此时的请求和用户最初的请求已经不属同一请求了

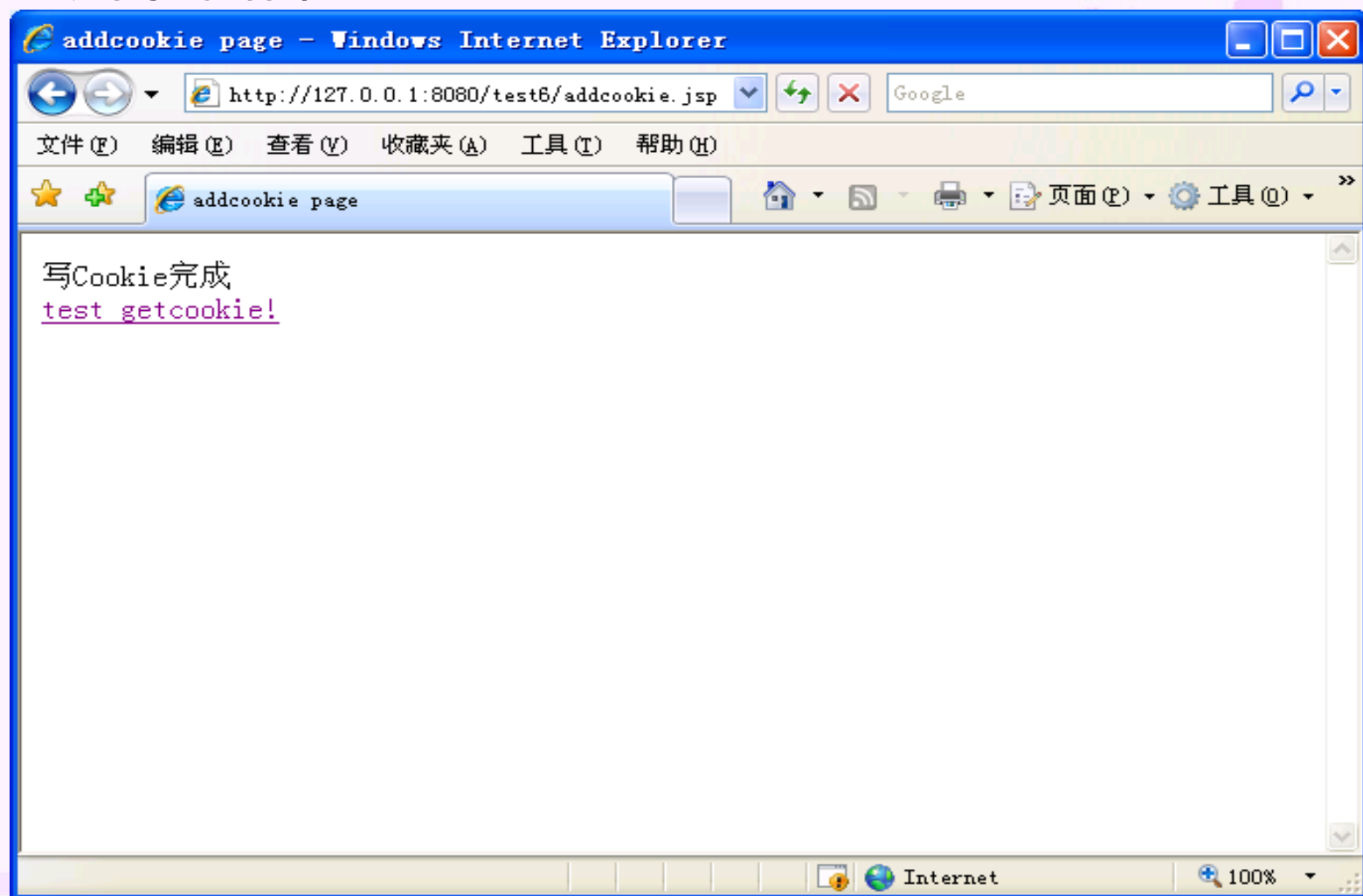
而转发是继续把当前的请求对象传递给转发后的页面，因此前后可以共用同一个请求对象。

重定向后在地址栏里最终显示的也是重定向以后页面的 URL，这也是它与转发的区别之一，转发后在地址栏里最终显示的是转发前页面的 URL。

用 response 对象实现向客户端写入 Cookie 的功能

```
<%@ page language="java"
contentType="text/html;charset=gb2312"%>
<html>
<head><title>addcookie page</title></head>
<body>
<%
Cookie cookie=new Cookie("username", "admin");
cookie.setMaxAge(2*60);
response.addCookie(cookie);
out.print(" 写 Cookie 完成 ");
%>
<br>
<a href="testcookie.jsp">test getcookie!</a>
</body>
</html>
```

访问的结果



在 testcookie.jsp 中，取到的 Cookie 对象 不为空，以取出 Cookie 中的用户名，一旦取到则停止循环。

```
<body>
<%
Cookie[] cookies = request.getCookies();
if(cookies==null){
    out.println(" 欢迎访问本网站!");
}else{
    for (int i = 0; i < cookies.length; i++) {
        if(cookies[i].getName().equals("username")) {
            out.println(cookies[i].getValue()+" 您好！欢迎再次访问本网站");
            break;
        }
    }
}
%>
</body>
```

在访问完 addcookie.jsp 后的 120 秒钟 Cookie 有效期内
点击图 6-18 中指向 testcookie.jsp 的超链接后，得到的
结果如图 6-19 所示：

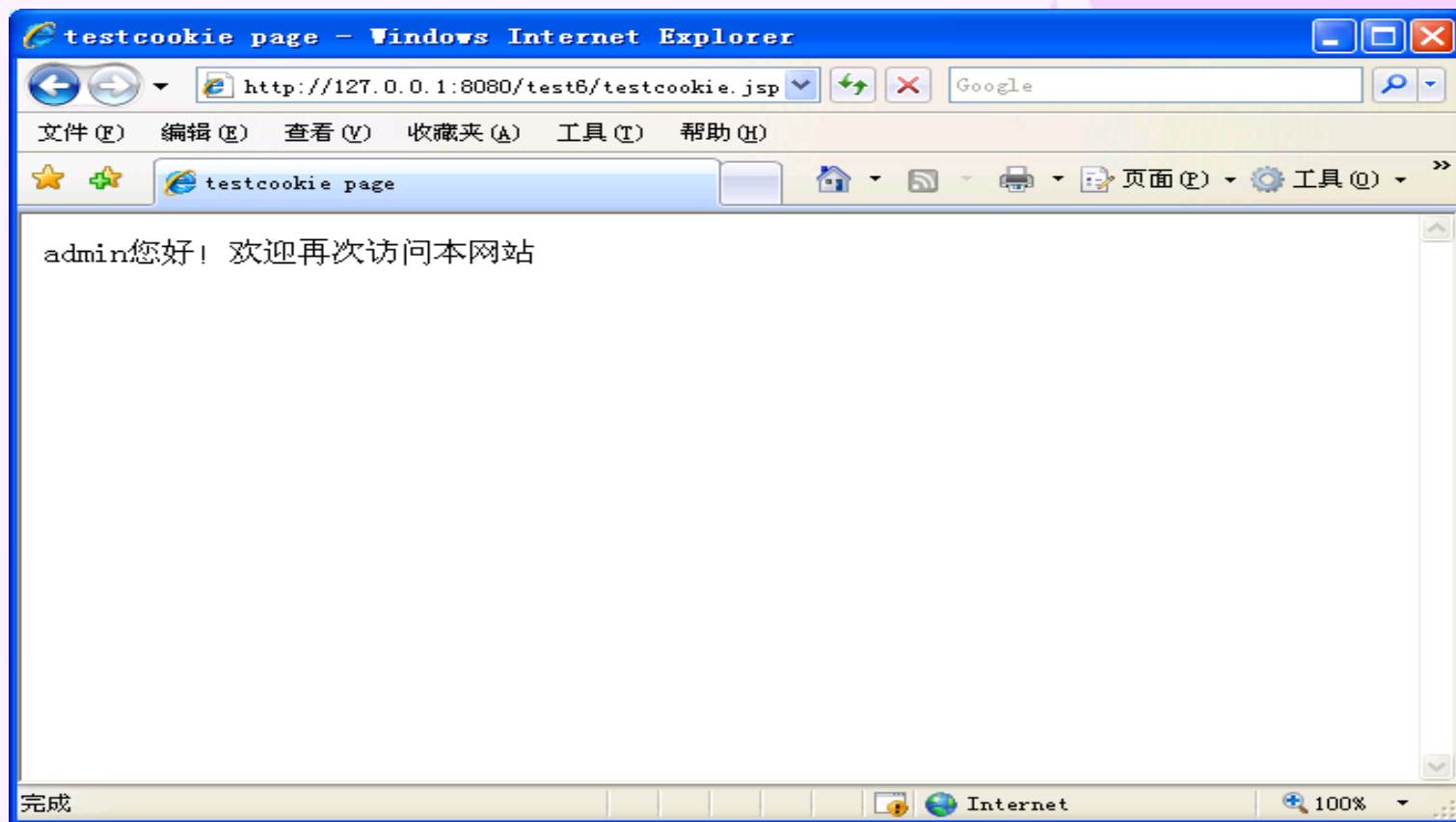
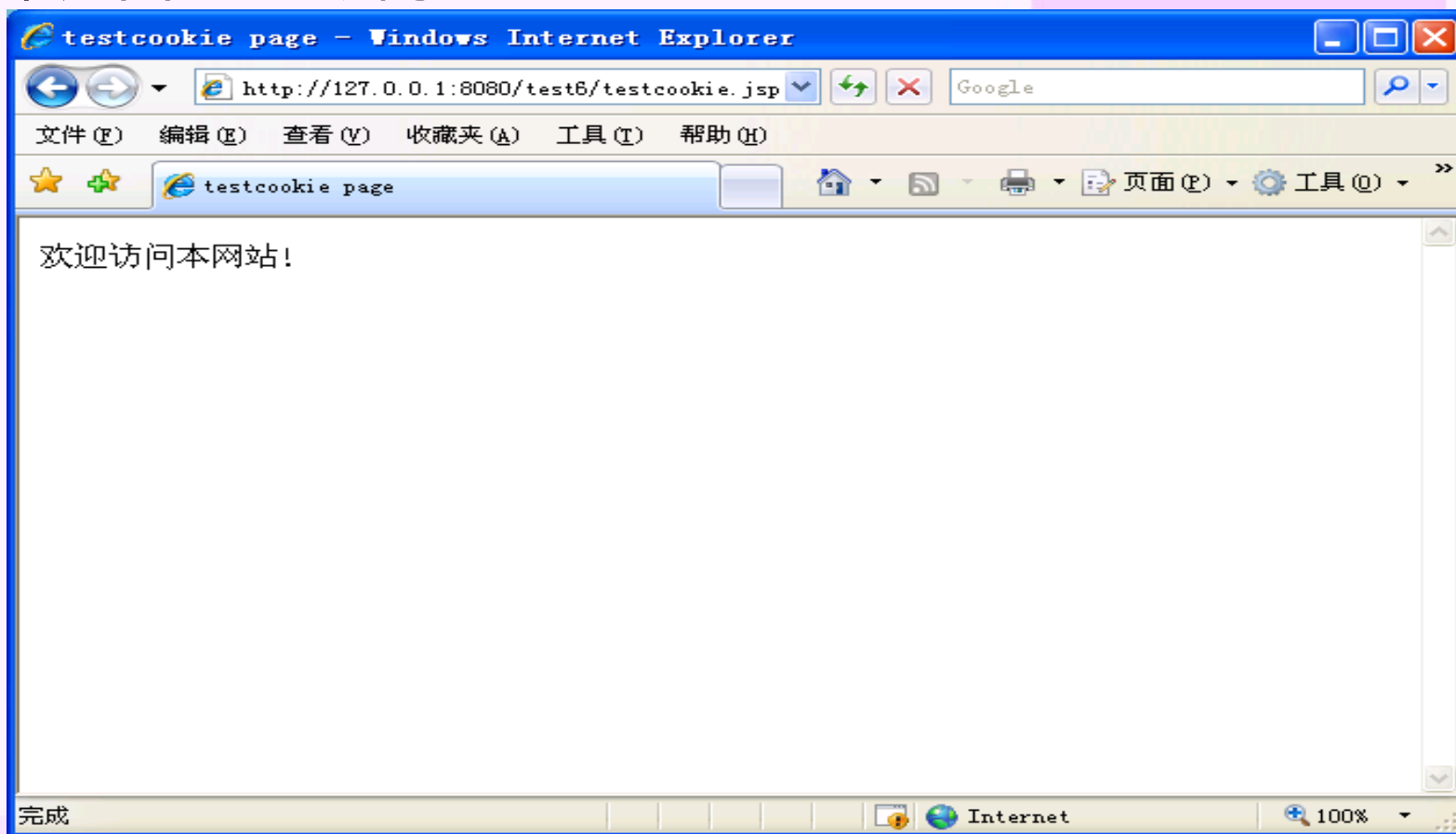


图 6-19 在 Cookie 有效期内访问 testcookie.jsp

关闭浏览器，再等待一段时间，等 Cookie 失效之后重新打开浏览器直接访问 testcookie.jsp，此时得到的结果如图 6-20 所示：



session 对象

session 对象是 `javax.servlet.http.HttpSession` 的实例。session 对象指的是客户端与服务器的会话，从客户访问服务器的一个 Web 应用开始，直到客户端与服务器断开连接为止。session 对象存在于服务器的内存中。

通常会话管理是通过服务器将 session ID 作为一个 cookie 存储在用户的 Web 浏览器中来唯一标识每个用户会话，session ID 的值是一个不会重复的字符串。

客户端与服务器的交互过程中，这个 session 对象一直存在，直到客户端关闭所有与该网站相关的网页后，这个 session 对象才被释放。

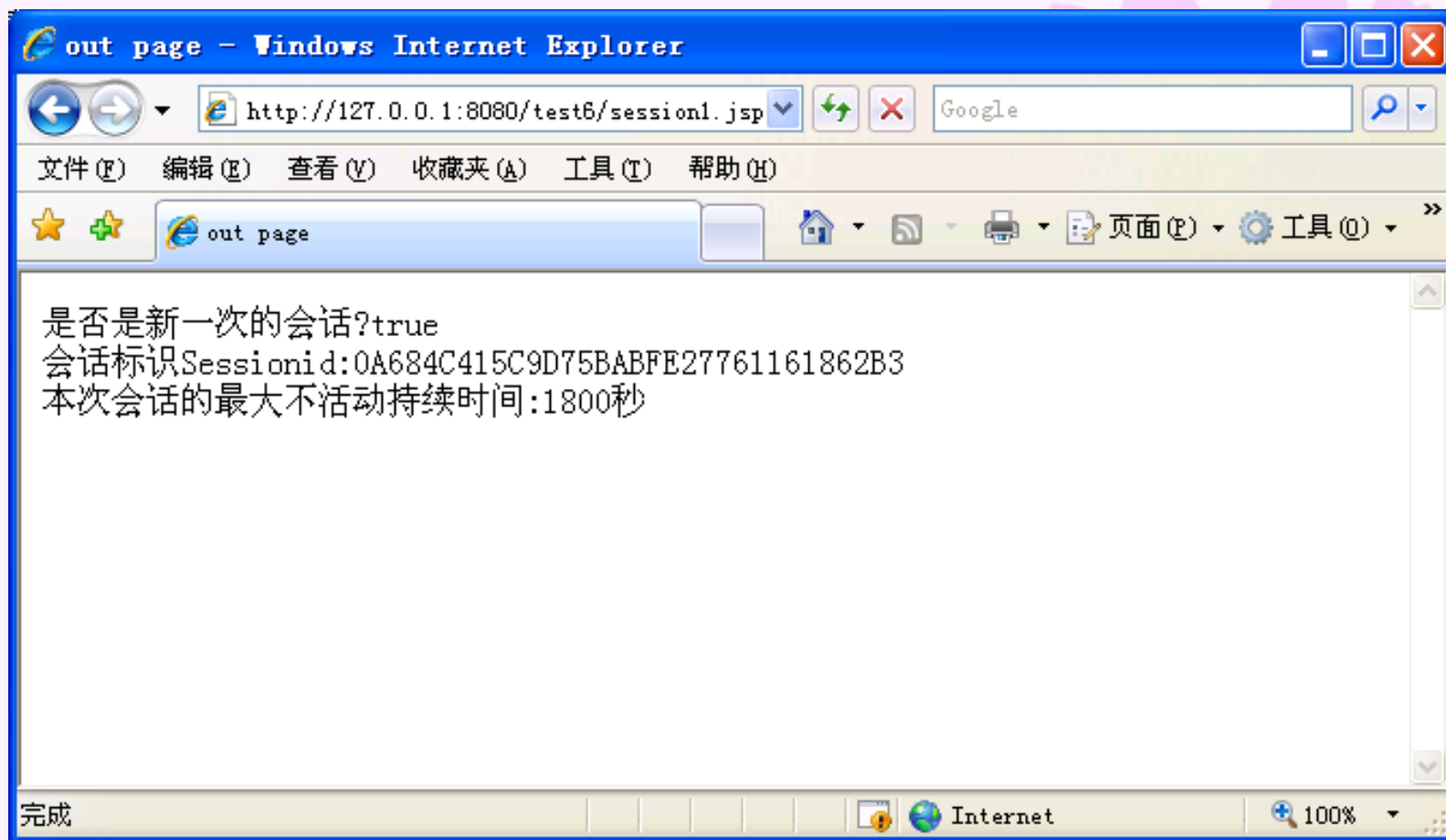
表 6-4 session 对象常用方法

方法名	功能
isNew	检查用户的会话对象是否是新创建的
getId	返回会话标识符，即 session ID
getMaxInactiveInterval	返回会话对象所允许的最大的不活动持续时间
setMaxInactiveInterval	设置会话对象所允许的最大的不活动持续时间
invalidate	使会话强制失效
setAttribute	设置会话中属性的值
getAttribute	返回会话中属性的值

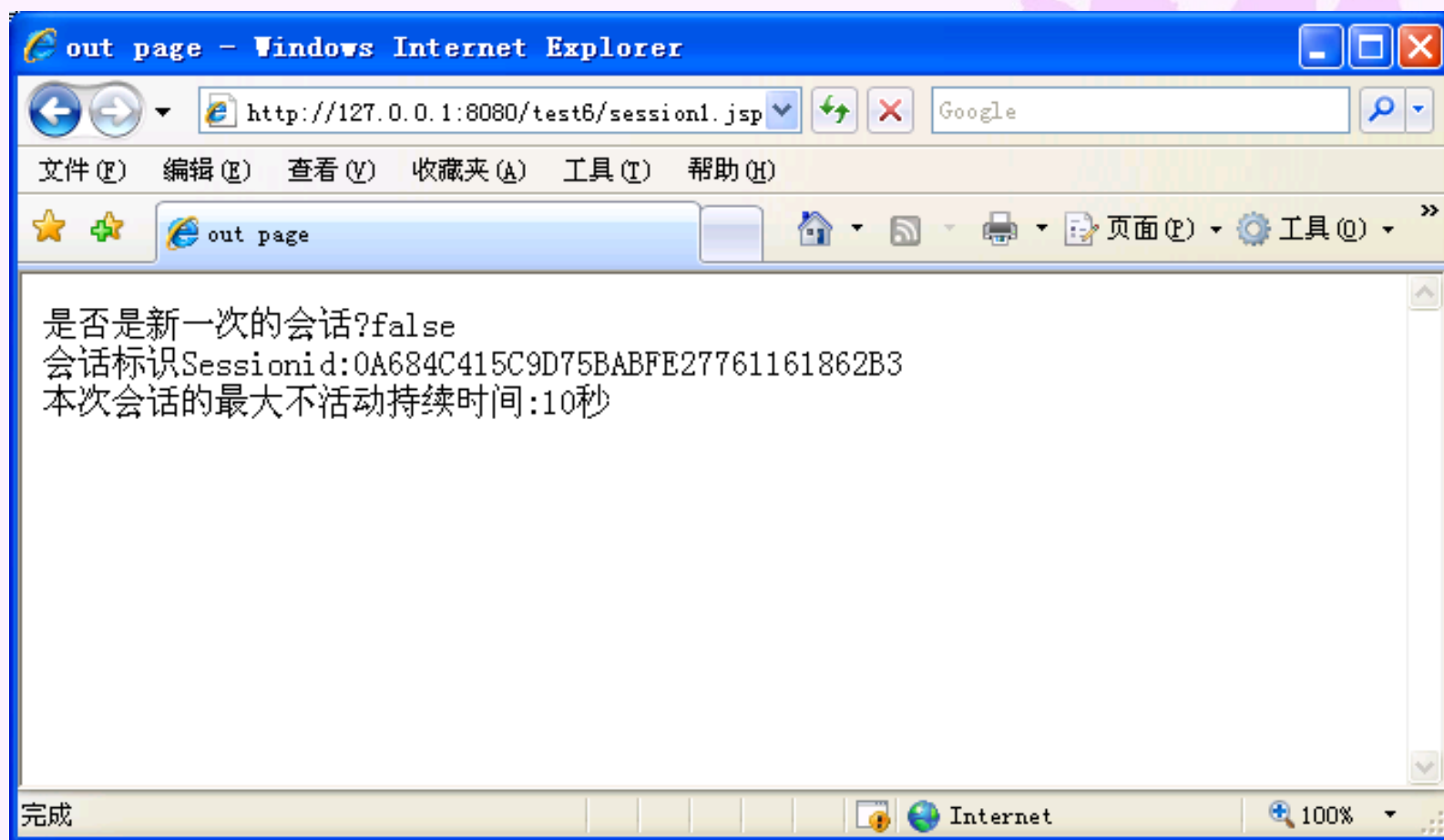
获取并设置 session 对象本身的各种信息。

```
<%@ page language="java"
contentType="text/html;charset=gb2312"%>
<html>
<head><title>session1 page</title></head>
<body>
是否是新一次的会话 ?<%=session.isNew()%><br>
会话标识 Sessionid:<%=session.getId()%><br>
本次会话的最大不活动持续时间 :<
%=session.getMaxInactiveInterval()%> 秒
<%session.setMaxInactiveInterval(10);%>
</body>
</html>
```

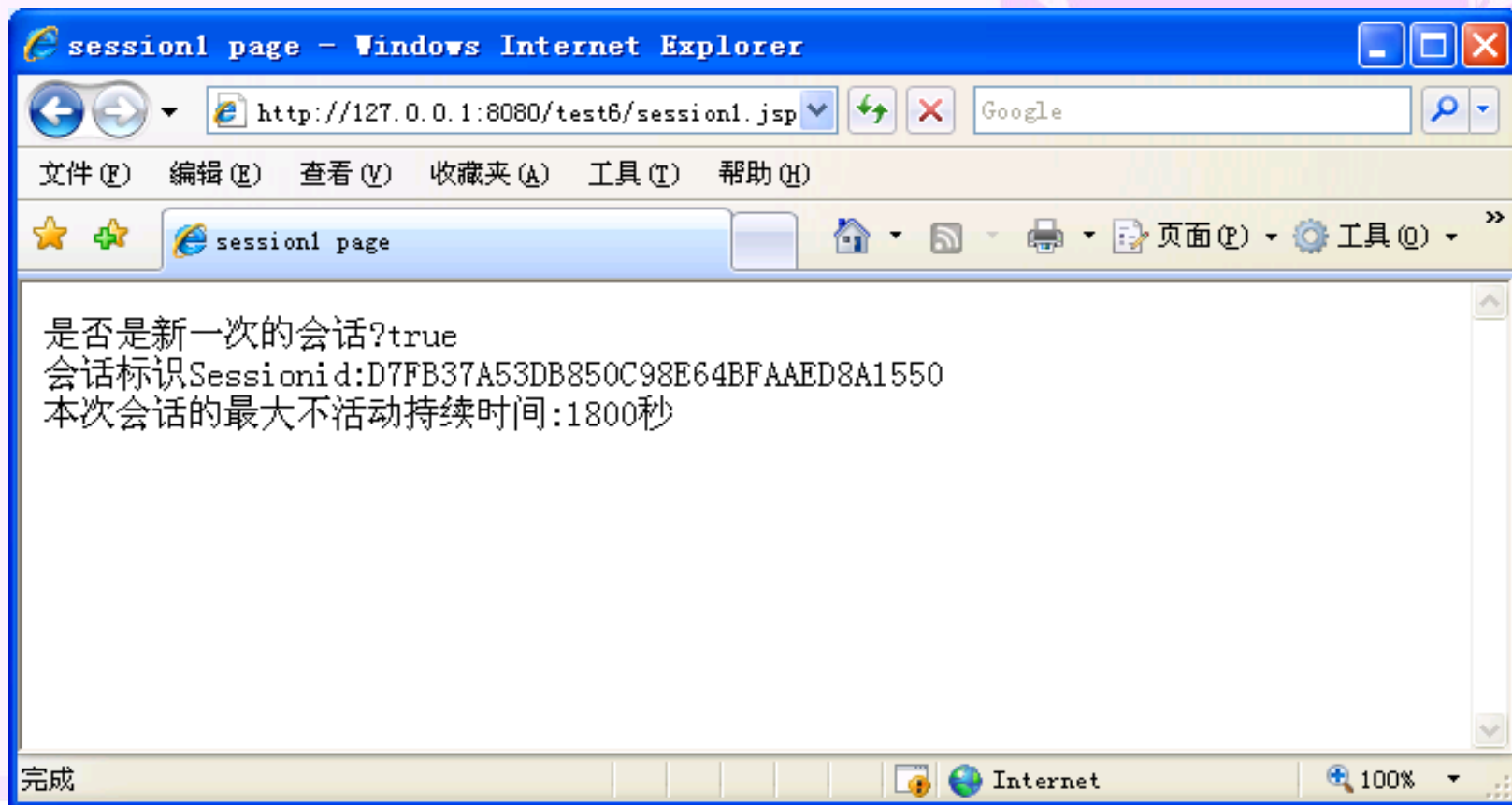
访问 session1.jsp 得到的结果如图 6-21 所示：



客户端在首次访问后的 10 秒之内再次刷新 session1.jsp，得到的结果如图 6-22 所示。



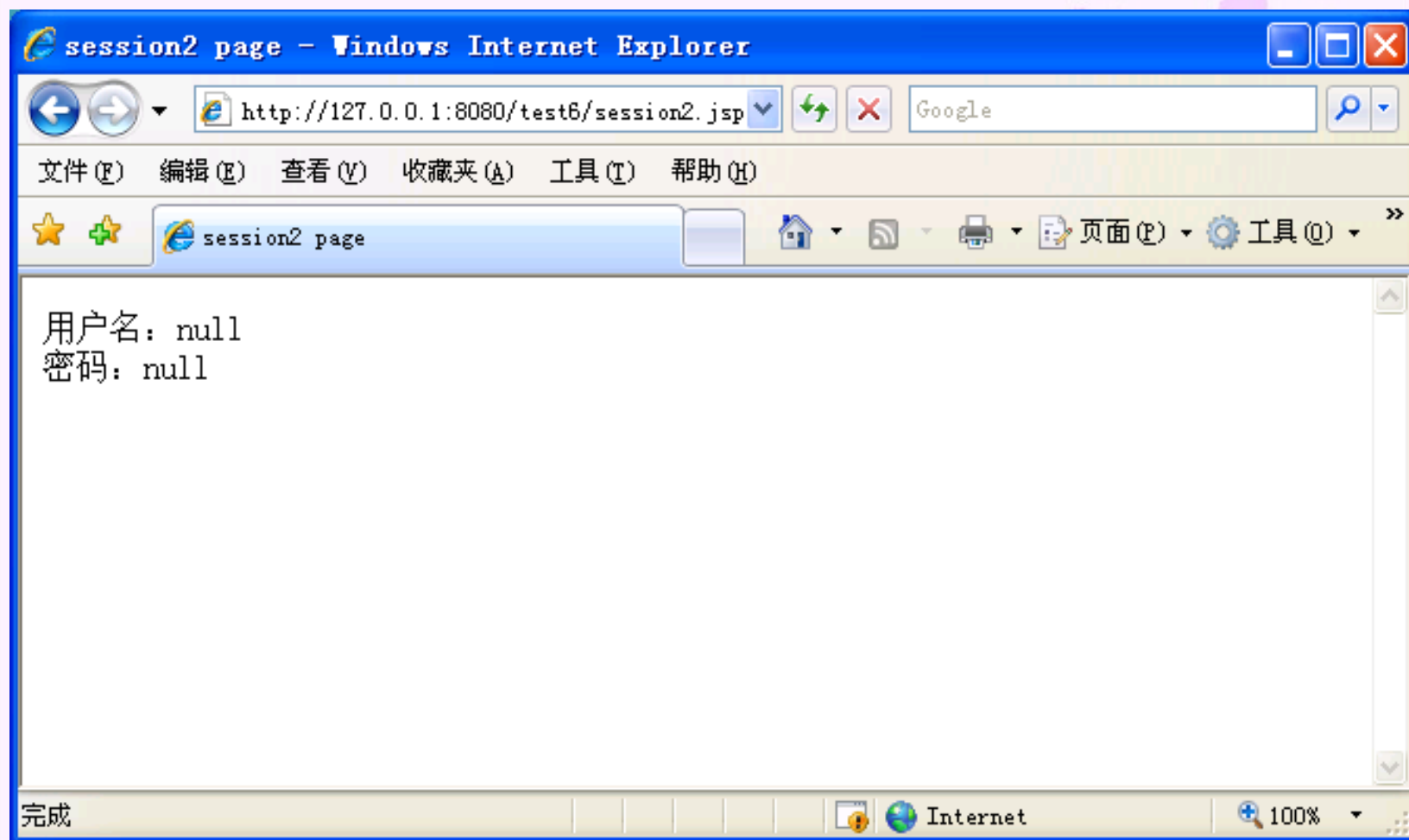
如果在首次访问后的 10 秒之后再次刷新 session1.jsp 或者关闭浏览器重新访问 session1.jsp，此时原先的 session 已经失效，得到的结果如图 6-23 所示。



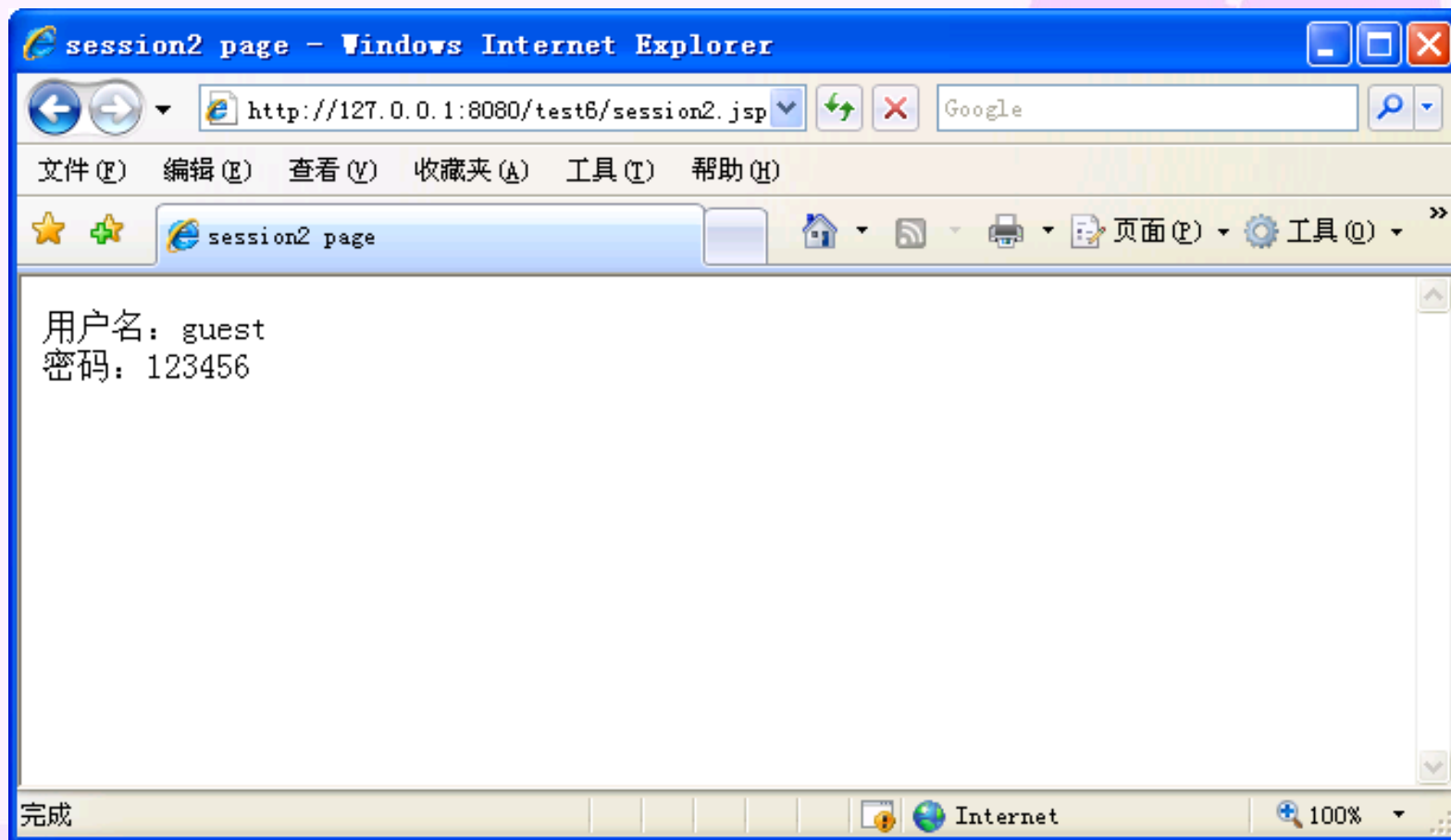
获取并设置 session 对象中的自定义属性。

```
<%@ page language="java"
contentType="text/html;charset=gb2312"%>
<html>
<head><title>session2 page</title></head>
<body>
用户名： <%=session.getAttribute("username")%>
密码： <%=session.getAttribute("password")%>
<%
session.setAttribute("username","guest");
session.setAttribute("password","123456");
%>
</body></html>
```

首次访问 session2.jsp 得到的结果如图 6-24 所示：



当 session2.jsp 中的 setAttribute 方法被执行后，我们再次刷新 session2.jsp，得到的结果如图 6-25 所示。



application 对象

application 对象是 `javax.servlet.ServletContext` 的实例。一个 Web 项目只有一个 application 对象，它开始于服务器的启动，直到服务器的关闭。在此期间，application 对象一直存在于服务器内存中。application 对象是所有用户共享的，可以存放公用的全局变量，与 application 对象相比，session 对象则是每个客户各自专有的，不同用户的 session 对象之间不可以共享和交互数据。

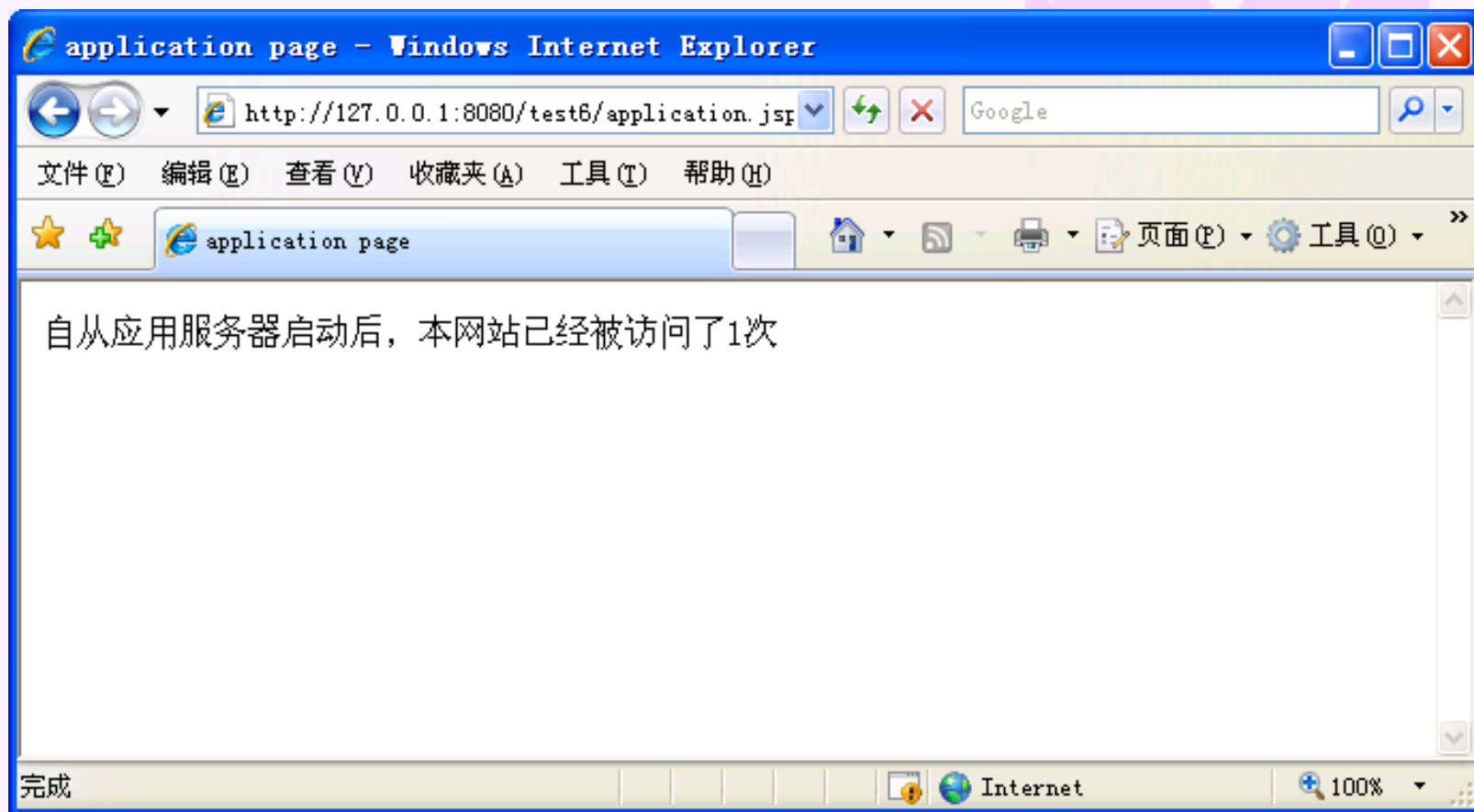
表 6-5 application 对象的常用方法

方法名	功能
setAttribute	设置某个 application 对象的某个属性值
getAttribute	以 Object 对象的形式返回对应名称的 application 对象的属性值

使用 application 对象实现网站计数器。

```
<body>
<%
Integer totalcount=1;
if(application.getAttribute("count")!=null)
{
    totalcount=(Integer)application.getAttribute("count")+
    1;
}
out.println(" 自从应用服务器启动后，本网站已经被访问
了 "+totalcount+" 次 ");
application.setAttribute("count",totalcount);
%>
</body>
```

打开浏览器首次访问该页面得到的结果如图 6-26 所示，不断的刷新该页面，计数值也会不断的增加。



page 对象

page 对象是 `java.lang.Object` 类型的，它是处理当前请求的 JSP 页的实现类的实例。page 对象的常用方法也是 `setAttribute` 和 `getAttribute`，使用方式与 `request`、`session` 以及 `application` 相似，只是所存储的属性其作用域仅仅是当前页面，一旦离开当前页面，page 中的属性也就随着 page 对象的结束而失效。

6.6 JSP 开发实例

在 Java Web 开发过程中，我们经常需要进行程序或页面间的转发。因为完成一个业务操作往往需要经历多个步骤，每一步骤完成处理后要转到下一个步骤。

转发的方式主要有两种：

- (1) 是使用请求转发器 (`RequestDispatcher`) 进行请求转发；
- (2) 是使用 `HttpServletResponse` 的 `sendRedirect` 方法，即响应重定向。

6.6.1 请求转发实例

RequestDispatcher 是请求转发器，RequestDispatcher 接口中定义了 forward 方法，它可以将当前的 request 对象转发到该 RequestDispatcher 指定的 Web 资源，可能是一个 Servlet 或者 JSP，然后由该资源文件继续处理请求并完成响应。

forward 方法不能通过 get 方式将参数附带在要转发到的资源路径的后面，但我们可以通过 request.setAttribute("username",username) 来储存参数并传至下一个资源文件，然后再通过 request.getAttribute("username") 来获取参数值。跳转后，服务器自动以下一页来回应客户端的请求，但在浏览器中的地址栏中不会看到下一页的 URL 地址。

实例简介

下面通过具体实例对请求转发进行介绍。

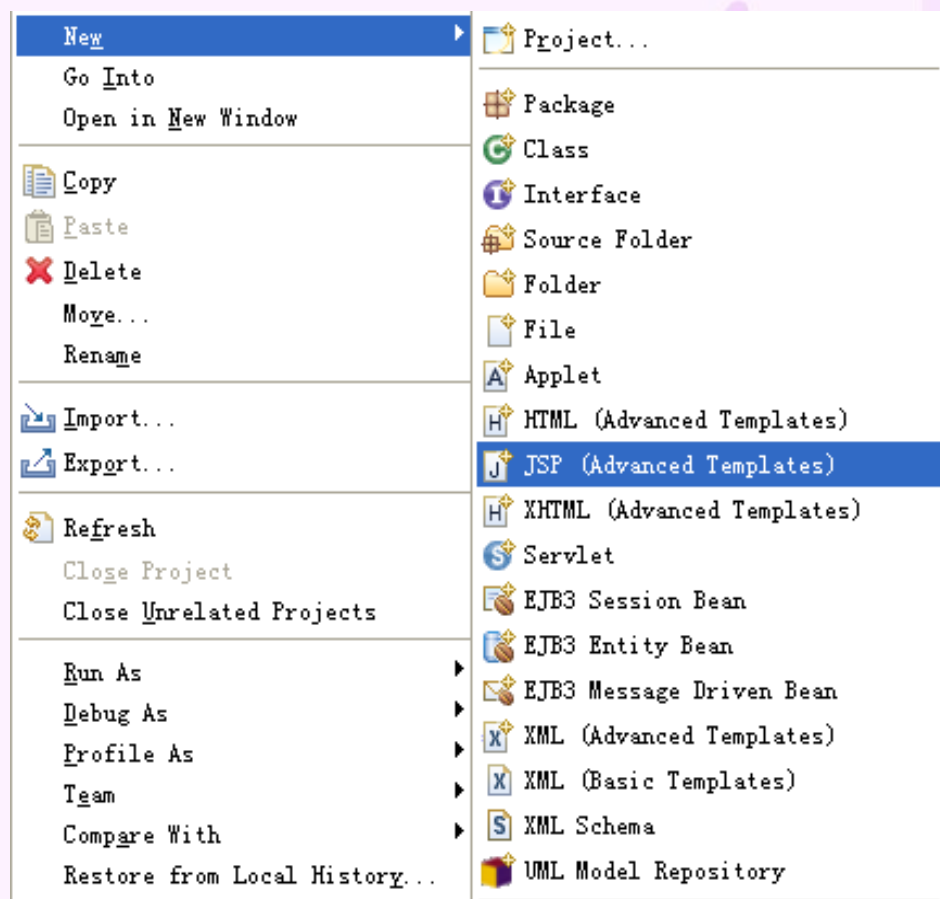
本例中有一个 JSP 注册页面，用户填写完注册信息后提交给一个 Servlet 文件进行处理，Servlet 文件接收数据后将请求转发到另一个 JSP 文件，最后由这个 JSP 文件将用户的注册信息打印出来。

具体步骤如下：

1. 在新建的项目中右键点击 WebRoot 文件夹，选择

【新建】→【JSP】
，新建一个名为“reguser1.jsp”的JSP页面。

如图所示：



reguser1.jsp 的主要代码如下：

```
<%@ page language="java" contentType="text/html; charset=gb2312"
%>
.....
<form method="post" action="reg" name="form1"
onsubmit="return check()">
.....
</form>
```

以上 JSP 文件的内容是由 HTML 文件直接修改而来，其中需要注意的是 <form> 标签的 method 属性和 action 属性。method 属性必须为 POST 方式，其含义是以 POST 方式将整个表单数据提交给服务器端的 Servlet 程序。

action 属性指定需要提交给哪一个 Servlet 的 URL。

需要注意的是：

通常我们更习惯采用 Dreamweaver 来设计 html，如果在 Dreamweaver 中设计好上述页面后，必须在整个文件前面添加如下代码：

```
<%@ page language="java"  
contentType="text/html;charset=gb2312" %>
```

完成以上修改后，将文件的后缀名修改为 JSP，然后直接用鼠标拖拽的方式导入到 MyEclipse 中。

2. 通过 MyEclipse 的 Servlet 创建向导新建一个名为 RegServlet 的 Servlet 文件，其“ Mapping URL” 设置为“ /reg”，作为该 Servlet 文件的访问路径。 web.xml 中的主要代码如下：

```
<servlet>
  <servlet-name>MyServlet</servlet-name>
  <servletclass>servlet.RegServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>MyServlet</servlet-name>
  <url-pattern>/reg</url-pattern>
</servlet-mapping>
```

RegServlet 文件中的核心处理代码：

(1) 参数的接收过程：

```
String username =  
StrConvert.ToChinese(request.getParameter("username"));  
String sex = StrConvert.ToChinese(request.getParameter("sex"));  
String[] interest = request.getParameterValues("interest");  
String interests = "";
```

(2) 参数的处理过程：

```
if (interest != null) {  
    for (int i = 0; i < interest.length; i++) {  
        interests = interests + StrConvert.ToChinese(interest[i]) + " ";  
    }  
    else  
        interests = " 没有任何爱好 ";
```

(3) 参数的转发

发

```
request.setAttribute("username", username);  
request.setAttribute("sex", sex);  
request.setAttribute("interests", interests);  
RequestDispatcher dispatcher =  
request.getRequestDispatcher  
("/reguser2.jsp");  
dispatcher.forward(request, response);
```

我们将 request 中的参数获取后，采用 StrConvert（与上一章中的相同）对其进行中文处理，并作为属性存到 request 对象里，最后请求转发到 reguser2.jsp。

reguser2.jsp 的主要代码如下：

```
.....  
    <td height="27" colspan="2" align="center">  
        <font size="4"> 您的注册信息 </font></td>  
.....  
    <td width="36%" height="43"><div align="left"> 您的用户名  
</td>  
    <td width="64%"><%=request.getAttribute("username")%></td>  
.....  
    <td height="39"> <div align="left"> 性别 :</div></td>  
    <td align="center"><%=request.getAttribute("sex")%></td>  
.....  
    <td height="52"><div align="left"> 个人爱好 : </div></td>  
    <td align="center"><%=request.getAttribute("interests")%>  
</td>  
.....
```

- 3. 部署完毕后，在地址栏中输入“http://localhost:8080/requestDis/reguser1.jsp”，得到如图 6-28 所示页面：

用户注册 - Microsoft Internet Explorer

文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)

后退 前进 停止 刷新 搜索 收藏夹 打印 打印范围 打印范围 打印范围

地址(Q) http://localhost:8080/servlet1/reguser1.jsp 转到 链接

新用户注册

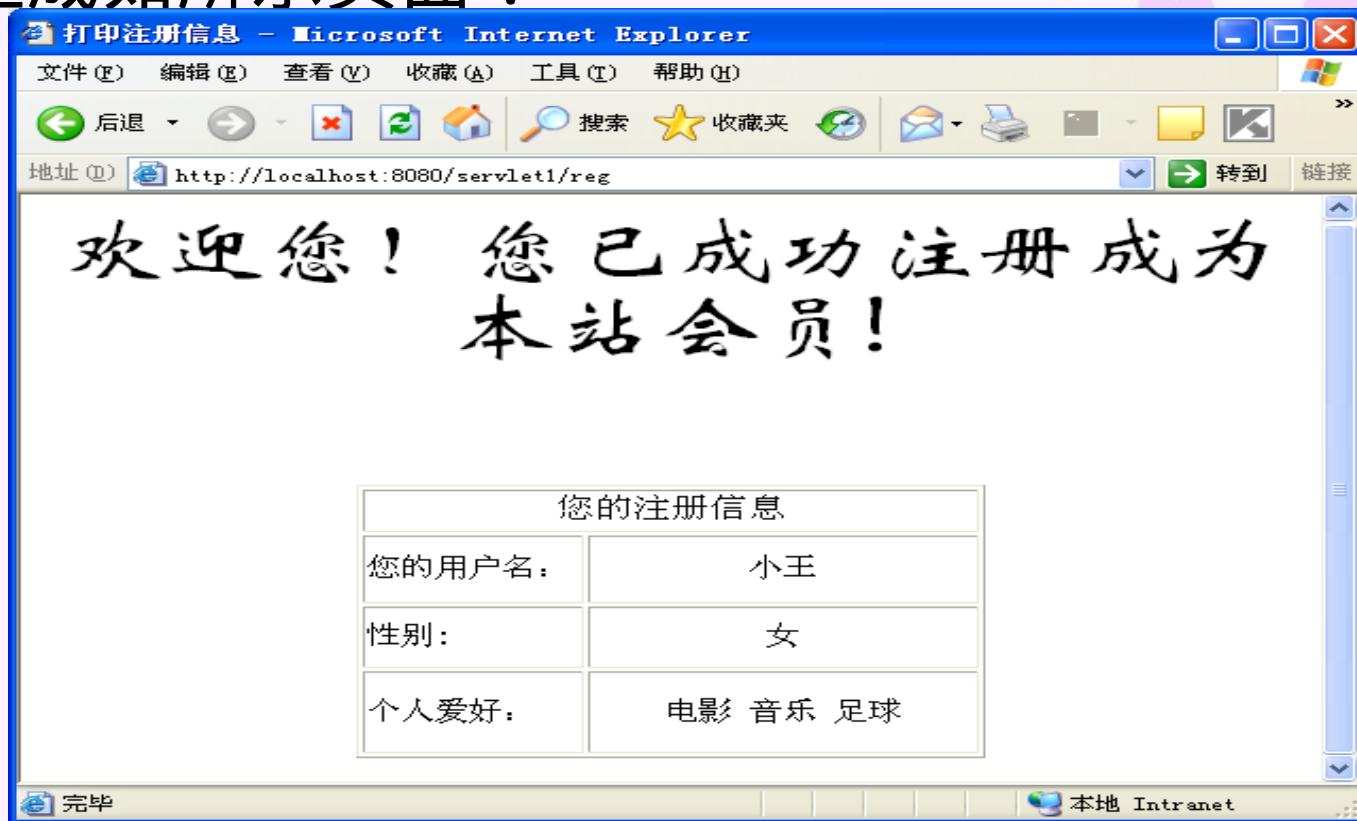
请准确的填写您的个人信息, 以便我们更好的为您服务, Thanks!

注册信息(注: 带*号为必填项)

用户名:	<input type="text" value="小王"/> *
密码:	<input type="password" value="*****"/> *
密码确认:	<input type="password" value="*****"/> *
性别:	女 <input type="button" value="v"/> *
个人爱好:	<input checked="" type="checkbox"/> 电影 <input checked="" type="checkbox"/> 音乐 <input type="checkbox"/> 编程 <input type="checkbox"/> 篮球 <input checked="" type="checkbox"/> 足球 <input type="checkbox"/> 其他

本地 Intranet

- 4. 填写完注册信息后，单击【提交】按钮，生成如所示页面：



采用 forward 方法进行页面跳转后，浏览器地址栏中的 URL 不会改变。

6.6.2 响应重定向实例

HttpServletResponse 接口定义了可用于页面重定向的 sendRedirect 方法，该方法的过程对于用户来说是透明的，浏览器会自动完成新的访问。

HttpServletResponse 的 sendRedirect 方法与 RequestDispatcher 接口的 forward 方法的效果类似，但两者存在以下区别：

1. HttpServletResponse 的 sendRedirect 方法是结束用户本次的请求重新向一个新的 Web 资源发送请求，而 RequestDispatcher 的 forward 方法则是继续将用户本次的请求转发给下一个 Web 资源。
2. HttpServletResponse 的 sendRedirect 方法不但可以在位于同一主机上的不同 web 应用之间进行重定向，而且可以重定向到其它服务器上的 web 资源，而 RequestDispatcher 的 forward 方法只能对所在 web 应用中的 web 资源进行转发。

3. HttpServletResponse 的 sendRedirect 方法可以带参数进行传递，而 RequestDispatcher 的 forward 方法则不可以传递参数。
例如：

```
Response.sendRedirect(reguser2.jsp?  
username="test");
```

4. 使用 HttpServletResponse 的 sendRedirect 重定向后，浏览器的地址栏里会显示重定向以后的 URL，而使用 RequestDispatcher 的 forward 方法后，浏览器的地址栏里仍然显示转发前的 URL。

下面通过具体实例对响应重定向进行介绍。

本实例的应用需求与采用请求转发方式实现的实例基本相同，即通过 JSP 注册页面将用户的注册信息提交给一个 Servlet 处理，不同的是，本例在 Servlet 中采用响应重定向的文件接收数据后将请求转发到另一个 JSP 文件，最后由这个 JSP 文件将用户的注册信息打印出来。

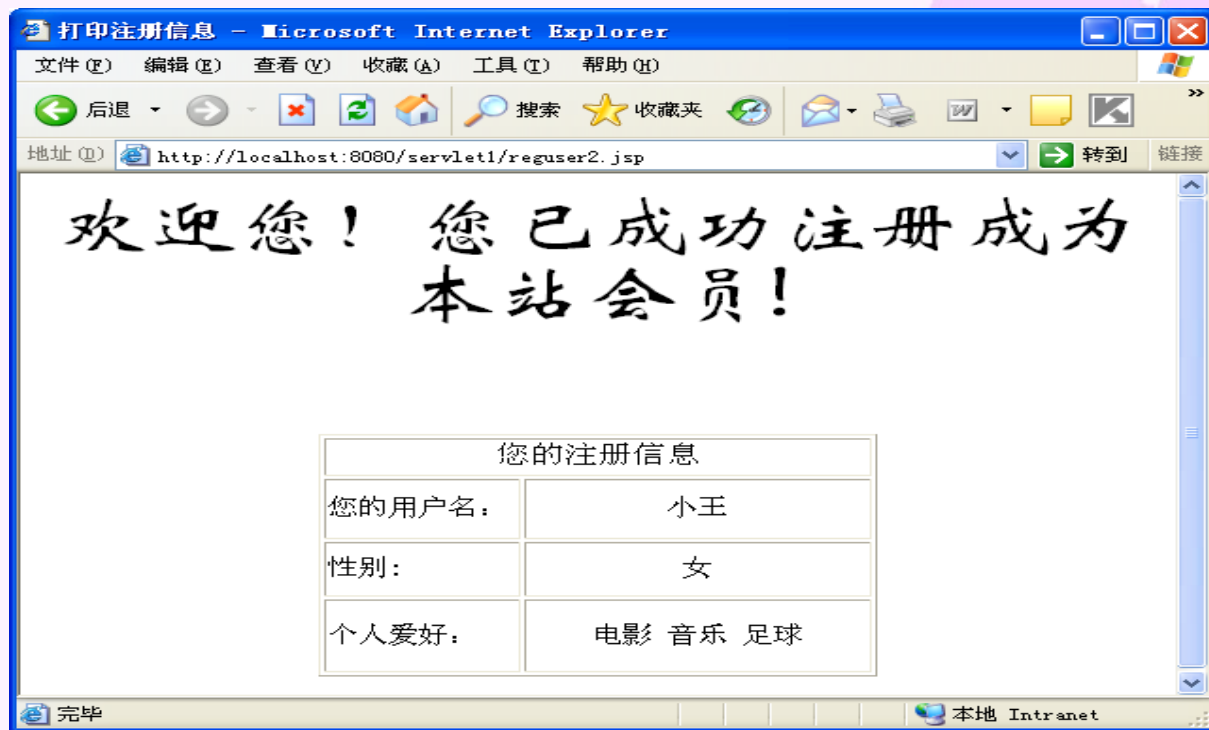
本例借助 session 对象，采用 HttpServletResponse 的 sendRedirect 方法重定向到 reguser2.jsp，修改 **RegServlet** 后的代码如下：

```
HttpSession session=request.getSession();  
session.setAttribute("username", username);  
session.setAttribute("sex", sex);  
session.setAttribute("interests", interests);  
response.sendRedirect("reguser2.jsp");
```

本例所需要开发两个 JSP 文件，分别为 reguser1.jsp 和 reguser2.jsp。其中，reguser1.jsp 为注册表单页，内容与上例中的相同，但 reguser2.jsp 文件则与上例有所不同，原先从 request 中获取属性值改为从 session 中获取，修改后的代码如下：

```
<td width="36%" height="43"><div align="left"> 您的用户名 :  
</div></td>  
<td width="64%" align="center">  
<%=session.getAttribute("username")%></td>  
.....  
<td height="39"> <div align="left"> 性别 :</div></td>  
<td align="center"><%=session.getAttribute("sex")%></td>  
.....  
<td height="52"><div align="left"> 个人爱好 : </div></td>  
<td align="center"><%=session.getAttribute("interests")%>  
</td>
```

项目部署完毕后，访问用户注册页面，填写完注册信息后，单击【提交】按钮，生成如图 6-30 所示页面：



用 `sendRedirect` 方法重定向后，地址栏不再是重定向之前的 URL，而是重定向以后的 URL

6.7 本章小结

本章介绍了 MVC 框架中视图层 JSP 的相关知识，主要包括 JSP 的工作原理、JSP 指令、JSP 脚本程序、JSP 动作以及 JSP 内置对象等，其中 JSP 的工作原理与上一章 Servlet 工作原理是紧密联系的。

在最后的开发实例中通过请求转发和响应重定向的两个实例详细介绍了如何将 JSP 与 Servlet 结合使用的应用技术。由于 JSP 的设计必须以静态页面 HTML 为基础，因此读者必须首先对 HTML 的设计熟练掌握。

本章习题

- 1、简述 JSP 的工作原理。
- 2、JSP 有哪些常用的指令？作用分别是什么？
- 3、JSP 的脚本程序主要包括哪几部分？请写出示例程序
- 4、JSP 有哪些常用的内置对象？作用分别是什么？请写出示例程序。
- 5、请求转发与响应重定向有何联系与区别？
- 6、请编写一个 Web 应用，要求如下：
 - ① 在 login.jsp 中设计包含用户名和密码的表单；
 - ② 在 check.jsp 中检验用户名和密码是否分别为“admin”和“123456”；
 - ③ 验证通过后将其用户名存到 session 中，并在 final.jsp 中取出打印；验证不通过则直接显示错误提示。